The thesis committee for Fangzhou Wei

Certifies that this is the approved version of the following thesis:

A Hybrid MPI/OpenMP Parallelization of the Adaptive Integral Method for Multi-Core Clusters

Approved by

Supervising Committee:

Ali E. Yilmaz, Supervisor

Mattan Erez

A Hybrid MPI/OpenMP Parallelization of the Adaptive Integral Method for Multi-Core Clusters

by

Fangzhou Wei, B.E.

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2009

Dedication

To my parents and my girl friend, Yizao

Acknowledgements

I would like to acknowledge my research adviser Professor Ali E. Yilmaz for his guidance, support and patience, for his scientific high standards.

I would like to thank Professor Yilmaz for the weekly research discussion slot for each individual student and his always available for research discussion.

I would like to thank Professor Erez for his patience and support.

<Dec. 5th 2009>

Abstract

A Hybrid MPI/OpenMP Parallelization of the Adaptive Integral Method for Multi-Core Clusters

Fangzhou Wei, M.S.E

The University of Texas at Austin, 2009

Supervisor: Ali E. Yilmaz

A hybrid of message passing and shared memory techniques is presented for scalable parallelization of the adaptive integral method (AIM), an FFT based algorithm, on clusters of identical multi-core processors. The proposed hybrid MPI/OpenMP parallelization scheme is based on a nested one-dimensional (1-D) slab decomposition of the 3-D auxiliary uniform grid and the associated AIM calculations: If there are M processors and T cores per processor, the scheme (i) divides the uniform grid into M slabs and MT sub-slabs, (ii) assigns each slab/sub-slab and the associated operations to one of the processors/cores, and (iii) uses MPI for inter-processor data communication and OpenMP for intra-processor data exchange. The MPI/OpenMP parallel AIM is used to accelerate the

MOM solution of combined-field integral equations pertinent to the analysis of scattering from perfectly conducting surfaces. The scalability and efficiency of the implementation are investigated theoretically and verified numerically by solving benchmark scattering problems on a (near) petaflop supercomputing cluster of quad-core processors. The timing and speedup results on up to 1024 processors show that the proposed hybrid MPI/OpenMP parallelization exhibits better strong scalability (fixed problem size speedup) compared to pure MPI parallelization when multiple cores are used on each processor.

Table of Contents

List of Tables

List of Figures

CHAPTER 1: INTRODUCTION

Traditional iterative method of moments (MOM) [1] solution of frequency-domain integral equations is limited to electrically small problems as it results in a dense-matrix equation, whose solution requires $O\left(N^2\right)$ operations per iteration and $O\left(N^2\right)$ bytes of memory for N degrees of freedom. Over the past two decades, several "fast algorithms" [2] have been developed to reduce the computational costs of MOM solvers, e.g., fast multi-pole method (FMM) [3-5] and fast Fourier transform (FFT) based accelerators [6-9] have been shown to reduce the MOM operation counts and memory requirements to $O\left(N\right)$ within logarithmic factors for broad classes of electrically large problems. To maximally harvest the available computational power, these fast algorithms are parallelized: A wide variety of MPI and OpenMP based parallel fast solvers have been developed that target distributed- and shared-memory architectures, respectively [6-14].

The recent emergence of multi-core architectures and the continuing deployment of supercomputing clusters of multi-core processors that result in a hierarchy of local and remote memory have blurred the shared-/distributed-memory distinction. MPI based parallel implementations of many algorithms that are traditionally scalable to a large number of processors suffer efficiency losses when deployed on such architectures and are often not "multi-core scalable", i.e., their parallel efficiency is significantly lower when multiple cores of the processors are used [15]. One avenue for overcoming this limitation is to use a hybrid MPI/OpenMP parallelization model [16-19] as has recently been presented for finite-difference time-domain solvers [20, 21], FMM algorithms [22], and direct MOM solvers accelerated by low-rank compression algorithms [23].

This thesis explores the parallel scalability of FFT accelerated iterative MOM solvers on multi-core clusters. Specifically, a typical pure message-passing (MPI) and a novel hybrid message-passing/shared-memory (MPI/OpenMP) technique are contrasted

for parallelizing the adaptive integral method (AIM) [4, 7], an FFT based algorithm for non-uniformly meshed structures. The hybrid method exploits the memory hierarchy of multi-core clusters via a nested parallelization scheme that uses MPI for inter-processor data communication and OpenMP for intra-processor (inter-core) data exchange. The AIM grid and the associated calculations including FFT operations are distributed among the cores by using a nested one-dimensional (1-D) slab decomposition. Both theoretical analysis and numerical results from benchmark electromagnetic scattering problems demonstrate that the resulting hybrid MPI/OpenMP parallelization of AIM outperforms a pure MPI version. All the simulations in this thesis are conducted on the Ranger [24] cluster at the Texas Advanced Computing Center (TACC), which is a near-petaflop cluster of 3936 computing nodes each of which consists of four quad-core processors and 8 GB of memory per processor (2 GB per core).

The rest of this thesis is organized as follows. Chapter 2 reviews three integral equation formulations commonly used in electromagnetic analysis and briefly formulates the MOM solution procedure and the AIM approach. It also introduces a set of scatterers that are used to benchmark different parallelization methods. Chapter 3 presents the MPI and hybrid MPI/OpenMP parallelization of iterative MOM solvers based on column and nested column-row decomposition, respectively. It analyses the parallel scalability of the two approaches and contrasts their performance on Ranger for the benchmark scatterers. Chapter 4 presents the two parallelization approaches for AIM and details the slab and the nested slab decomposition used for its MPI and hybrid MPI/OpenMP parallelization, respectively. A detailed scalability analysis and extensive numerical results on Ranger quantify the multi-core scalability of the two approaches. Chapter 5 concludes the thesis and discusses future research directions.

CHAPTER 2: THE ADAPTIVE INTEGRAL METHOD

This chapter briefly reviews (i) the electric, magnetic, and combined field integral equation (EFIE, MFIE, and CFIE) pertinent to the analysis of time-harmonic scattering from perfectly conducting surfaces residing in an unbounded homogenous medium [25, 26], (ii) the MOM solution of these integral equations, and (iii) the AIM acceleration. It then presents a set of scatterers that are used to verify that the computational costs of the MOM and AIM implementations agree with the theoretical expectations. These scatterers are also used for benchmarking parallel implementations throughout the thesis.

2.1 Integral Equations

Consider an arbitrarily shaped 3-D perfectly electrically conducting (PEC) scatterer with surface S that resides in an unbounded, lossless, homogeneous dielectric medium with permittivity ε and permeability μ (Fig. 2.1).

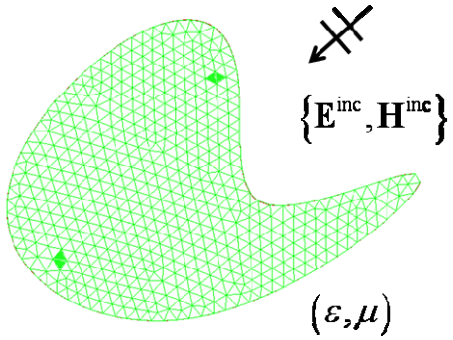


Figure 2.1: Scattering from a PEC scatterer meshed with triangular patches.

A time-harmonic incident electromagnetic field $\left\{\mathbf{E}^{\text{inc}}, \mathbf{H}^{\text{inc}}\right\}$ induces a surface current on the scatterer that in turn generates the scattered field $\left\{\mathbf{E}^{\text{sca}}, \mathbf{H}^{\text{sca}}\right\}$, which can be expressed in terms of the vector and scalar potentials $\mathbf{A}$ and Φ as ($e^{j\omega t}$ time variation is assumed and suppressed)

$$\left\{\mathbf{E}^{\text{sca}}\left(\mathbf{r}\right), \mathbf{H}^{\text{sca}}\left(\mathbf{r}\right)\right\} = \left\{-j\omega\mathbf{A}\left(\mathbf{r}\right) - \nabla\Phi\left(\mathbf{r}\right), \frac{\nabla \times \mathbf{A}\left(\mathbf{r}\right)}{\mu}\right\} \tag{2.1}$$

where

$$\left\{\mathbf{A}\left(\mathbf{r}\right),\Phi\left(\mathbf{r}\right)\right\}=\left\{\iint_S \mu\mathbf{J}\left(\mathbf{r}'\right)g\left(R\right)ds',-\frac{1}{j\omega\varepsilon}\iint_S \nabla'\cdot\mathbf{J}\left(\mathbf{r}'\right)g\left(R\right)ds'\right\} \qquad (2.2)$$
$$g\left(R\right)=e^{-jkR}\Big/4\pi R$$

Here, $\mathbf{J}$ is the surface current density, $R=\left|\mathbf{r}-\mathbf{r}'\right|$ is the distance between source point $\mathbf{r}'$ and observation point $\mathbf{r}$, and $k=\omega\sqrt{\mu\varepsilon}$ is the wave number. Enforcing tangential boundary conditions at the PEC surface yields the frequency domain EFIE and MFIE. The CFIE is a linear combination of them:

$$-\hat{\mathbf{n}}\left(\mathbf{r}\right)\times\left(\hat{\mathbf{n}}\left(\mathbf{r}\right)\times\mathbf{E}^{\text{inc}}\left(\mathbf{r}\right)\right)=\hat{\mathbf{n}}\left(\mathbf{r}\right)\times\left(\hat{\mathbf{n}}\left(\mathbf{r}\right)\times\mathbf{E}^{\text{sca}}\left(\mathbf{r}\right)\right) \qquad \left(\text{EFIE}\right)$$
$$\hat{\mathbf{n}}\left(\mathbf{r}\right)\times\mathbf{H}^{\text{inc}}\left(\mathbf{r}\right)=\mathbf{J}\left(\mathbf{r}\right)-\hat{\mathbf{n}}\left(\mathbf{r}\right)\times\mathbf{H}^{\text{sca}}\left(\mathbf{r}\right) \qquad \left(\text{MFIE}\right) \qquad (2.3)$$
$$\text{CFIE}=\alpha\text{EFIE}+\eta\left(1-\alpha\right)\text{MFIE} \qquad \left(\text{CFIE}\right)$$

Here, $0\le\alpha\le1$ and $\hat{\mathbf{n}}$ is the outward directed unit vector normal to S.

2.2 Method of Moments (MOM)

To numerically solve the integral equations, the MOM is applied. First, the current density is expanded as

$$\mathbf{J}\left(\mathbf{r}\right)\cong\sum_{k'=1}^{N}I_{k'}\mathbf{S}_{k'}(\mathbf{r}) \qquad (2.4)$$

Here, $\mathbf{S}_{k'}$ denotes the k'^{th} RWG basis function defined on pairs of triangular patches [27], $I_{k'}$ is the unknown current coefficient associated with $\mathbf{S}_{k'}$, and N is the number of unknowns. Second, the integral equations are converted into matrix equations via Galerkin testing, which uses the same testing functions as the basis functions. This results in the MOM matrix equation

$$\mathbf{Z}_{N\times N}\mathbf{I}_{N\times1}=\mathbf{V}_{N\times1}^{\text{inc}} \qquad (2.5)$$

Here, $\mathbf{Z}$ is the impedance matrix, $\mathbf{V}^{\text{inc}}$ is the tested incident field vector, and $\mathbf{I}$ is the current coefficient vector. For the CFIE, the entries of the matrices and vectors are given as (for $k'=1,...,N$ and $k=1,...,N$)

$$\mathbf{Z}[k,k'] = \alpha \left(\begin{array}{l} j\omega\mu \iint_S \iint_S \mathbf{S}_k(\mathbf{r}) \cdot \mathbf{S}_{k'}(\mathbf{r}') g(R)\, ds' ds \\ + \dfrac{1}{j\omega\varepsilon} \iint_S \iint_S \nabla \cdot \mathbf{S}_k(\mathbf{r}) \nabla' \cdot \mathbf{S}_{k'}(\mathbf{r}') g(R)\, ds' ds \end{array} \right)$$

$$+\eta(1-\alpha) \left(\begin{array}{l} \iint_S \mathbf{S}_k(\mathbf{r}) \cdot \mathbf{S}_{k'}(\mathbf{r})/2\, ds \\ + \iint_S \mathbf{S}_k(\mathbf{r}) \cdot \hat{\mathbf{n}}(\mathbf{r}) \times \iint_{P.V.S} \mathbf{S}_{k'}(\mathbf{r}') \times \nabla g(R)\, ds' ds \end{array} \right) \tag{2.6}$$

$$\mathbf{I}[k'] = I_{k'}$$

$$\mathbf{V}^{\text{inc}}[k] = \iint_S \alpha \mathbf{S}_k(\mathbf{r}) \cdot \mathbf{E}^{\text{inc}}(\mathbf{r}) + \eta(1-\alpha) \mathbf{S}_k(\mathbf{r}) \cdot \hat{\mathbf{n}}(\mathbf{r}) \times \mathbf{H}^{\text{inc}}(\mathbf{r})\, ds$$

where P.V.S stands for the Cauchy principle value. The EFIE and MFIE matrices and vectors can be deduced from the above equations by setting α to 1 and 0, respectively. Third, and finally, the desired current coefficients are found by solving the matrix equation (2.5). Both direct and iterative solution techniques are widely used for solving the MOM matrix equation [28, 29]. In this thesis, a TFQMR [30] iterative solver is adopted.

The iterative MOM solution procedure requires $O\left(N^2\right)$ operations to fill the dense $N \times N$ impedance matrix, $O\left(N^2\right)$ memory to store it, and $O\left(N^2\right)$ operations to multiply it with trial vectors at each iteration. Several classes of fast algorithms have been developed to accelerate the MOM procedure [2].

2.3 Adaptive Integral Method (AIM)

AIM [6] is an FFT based algorithm that accelerates the iterative MOM solution of integral equations for arbitrarily shaped scatterers by exploiting the translational invariance of the integral kernel through an auxiliary uniform grid (Fig. 2.2). The number of points on the auxiliary grid is denoted by $N_{\text{C}} = N_{cx} \times N_{cy} \times N_{cz}$, where N_{cx}, N_{cy}, and N_{cz} are the number of auxiliary grid points along x, y, and z dimensions, respectively.

For general 3-D scatterers $N_{\mathrm{C}} \sim N^{1.5}$ and for quasi-planar surfaces that have much larger transverse dimensions than their height, $N_{\mathrm{C}} \sim N$.

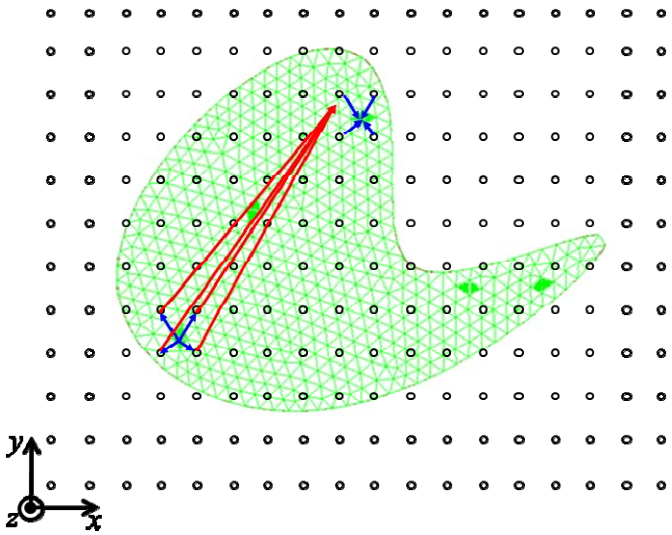


Figure 2.2: A pictorial description of the AIM projection, propagation, and interpolation steps. The circles show the auxiliary uniform grid points.

Using the auxiliary grid, the AIM separates the MOM matrix entries into "far-zone" and "near-zone" terms. Then, at each iteration, it (i) projects currents onto the auxiliary grid, (ii) propagates fields onto the same grid using FFTs, (iii) interpolates fields onto the scatterer mesh from the grid, and (iv) corrects near-zone terms. This four-step algorithm can be expressed in matrix form as follows. AIM approximates the MOM matrix as

$$\mathbf{Z}_{N\times N} \approx \mathbf{Z}^{\mathrm{near}}_{N\times N} + \mathbf{Z}^{\mathrm{FFT}}_{N\times N} \tag{2.7}$$

where

$$\begin{aligned}\mathbf{Z}^{\mathrm{FFT}}_{N\times N} = &\left(1-\alpha\right)\begin{bmatrix}\Gamma^{\mathrm{T}}_x & \Gamma^{\mathrm{T}}_y & \Gamma^{\mathrm{T}}_z\end{bmatrix}\begin{bmatrix}0 & \mathbf{G}^{z}_{N_{\mathrm{C}}\times N_{\mathrm{C}}} & -\mathbf{G}^{y}_{N_{\mathrm{C}}\times N_{\mathrm{C}}}\\ -\mathbf{G}^{z}_{N_{\mathrm{C}}\times N_{\mathrm{C}}} & 0 & \mathbf{G}^{x}_{N_{\mathrm{C}}\times N_{\mathrm{C}}}\\ \mathbf{G}^{y}_{N_{\mathrm{C}}\times N_{\mathrm{C}}} & -\mathbf{G}^{x}_{N_{\mathrm{C}}\times N_{\mathrm{C}}} & 0\end{bmatrix}\begin{bmatrix}\Lambda_x\\ \Lambda_y\\ \Lambda_z\end{bmatrix}\\ &+\alpha\begin{bmatrix}\Lambda^{\mathrm{T}}_x & \Lambda^{\mathrm{T}}_y & \Lambda^{\mathrm{T}}_z\end{bmatrix}\begin{bmatrix}\mathbf{G}^{\mathrm{A}}_{N_{\mathrm{C}}\times N_{\mathrm{C}}}\Lambda_x\\ \mathbf{G}^{\mathrm{A}}_{N_{\mathrm{C}}\times N_{\mathrm{C}}}\Lambda_y\\ \mathbf{G}^{\mathrm{A}}_{N_{\mathrm{C}}\times N_{\mathrm{C}}}\Lambda_z\end{bmatrix}+\alpha\Lambda^{\mathrm{T}}_{\nabla}\mathbf{G}^{\phi}\Lambda_{\nabla}\end{aligned} \tag{2.8}$$

Here, $\Lambda_{x,y,z,\nabla}$ and $\Gamma_{x,y,z}$ are $N_{\mathrm{C}} \times N$ sparse matrices that store the mapping coefficients for projecting currents from the surface mesh to the auxiliary grid and the superscript T denotes the matrix transpose. These mapping coefficients relate the basis functions to the associated auxiliary grid points and are found by the moment matching scheme [6], i.e., by matching the multi-pole moments of the auxiliary sources using the mapping coefficients Λ_x, Λ_y, Λ_z, Λ_∇, Γ_x, Γ_y, and Γ_z to those of the functions $\hat{\mathbf{x}} \cdot \mathbf{S}_k$, $\hat{\mathbf{y}} \cdot \mathbf{S}_k$, $\hat{\mathbf{z}} \cdot \mathbf{S}_k$, $\nabla \cdot \mathbf{S}_k$, $\hat{\mathbf{x}} \cdot \hat{\mathbf{n}} \times \mathbf{S}_k$, $\hat{\mathbf{y}} \cdot \hat{\mathbf{n}} \times \mathbf{S}_k$, and $\hat{\mathbf{z}} \cdot \hat{\mathbf{n}} \times \mathbf{S}_k$, respectively (In all the results in this thesis, up to third order moments are matched). The $\mathbf{G}^{x,y,z,\mathbf{A},\phi}$ matrices, also referred to as the "Green function kernel", are dense (3-level) block-Toeplitz matrices and can be multiplied efficiently via multidimensional FFTs. Their entries are:

$$\begin{aligned} &\{\mathbf{G}^x\left[u,u'\right],\mathbf{G}^y\left[u,u'\right],\mathbf{G}^z\left[u,u'\right],\mathbf{G}^{\mathbf{A}}\left[u,u'\right],\mathbf{G}^\phi\left[u,u'\right]\} = \\ &\{\partial_x,\partial_y,\partial_z,j\omega\mu,\frac{1}{j\omega\varepsilon}\}\frac{e^{-jk\left|\mathbf{r}_u-\mathbf{r}_{u'}\right|}}{4\pi\left|\mathbf{r}_u-\mathbf{r}_{u'}\right|} \end{aligned} \tag{2.9}$$

Here, u and u' are the observer and source grid point position on the auxiliary grid, respectively, and $\mathbf{G}^{x,y,z,\mathbf{A},\phi}\left[u,u'\right]$ are set to zero to avoid singularities.

In (2.7), the $\mathbf{Z}^{\mathrm{near}}$ matrix, which is a sparse matrix with N_{near} nonzero entries, corrects near-zone errors:

$$\mathbf{Z}^{\mathrm{near}}\left[k,k'\right] = \begin{cases} 0, & \text{if } R_{k,k'} > \gamma\Delta c \\ \mathbf{Z}\left[k,k'\right] - \mathbf{Z}^{\mathrm{FFT}}\left[k,k'\right], & \text{if } R_{k,k'} < \gamma\Delta c \end{cases} \tag{2.10}$$

Here, $R_{k,k'}$ is the minimum "grid distance" [6, 9] between the auxiliary points associated with the basis and testing functions, Δc is the grid spacing, and γ is a parameter that sets the near-zone region. In practice, Δc is comparable to the edge lengths on the scatterer mesh and is typically 1/20th-1/6th of the wavelength, $1 \le \gamma \le 6$, and $N_{\mathrm{near}} \sim N$ when the structure of interest is electrically large and devoid of geometrical details.

The AIM acceleration has the following computational costs: AIM requires $O\left(N\right)$ operations to fill $\Lambda_{x,y,z,\nabla}$ and $\Gamma_{x,y,z}$ matrices, $O\left(N_{\mathrm{C}}\right)$ operations to fill the $\mathbf{G}^{x,y,z,\mathbf{A},\phi}$ matrices, and $O\left(N_{\mathrm{near}}\right)$ operations to fill the $\mathbf{Z}^{\mathrm{near}}$ matrix. Thus, it requires $O\left(N_{\mathrm{near}}+N_{\mathrm{C}}\right)$ operations and $O\left(N_{\mathrm{near}}+N_{\mathrm{C}}\right)$ bytes of storage space to fill and store the matrices. In the iterative solution stage, at each iteration, AIM requires $O\left(N\right)$ operations for the projection and interpolation steps, $O\left(N_{\mathrm{near}}\right)$ operations for the near-zone correction step, and $O\left(N_{\mathrm{C}}\log N_{\mathrm{C}}\right)$ operations for the propagation step. Thus, it requires $O\left(N_{\mathrm{near}}+N_{\mathrm{C}}\log N_{\mathrm{C}}\right)$ operations per iteration.

2.4 Computational Complexity Validation

The computational complexity of the MOM and AIM implementations are verified next. Different sizes of PEC square plates and spheres are used here and throughout the thesis as benchmark numerical examples. EFIE and CFIE (with $\alpha=0.6$) formulations are used to analyze scattering from plates and spheres, respectively. These scatterers represent the best- and worst-case extremes for AIM as N_{C} is proportional to N for plates and to $N^{1.5}$ for spheres. The edge length L_{P} of the plates and the radius L_{S} of the spheres are scaled from λ to 256λ and λ to 64λ, respectively. The relative root-mean-square error in the VV-polarized bistatic radar cross section (RCS) $\sigma_{\theta\theta}$ is computed for measuring the accuracy and is denoted by $err_{\theta\theta}$:

$$err_{\theta\theta}=\left(\frac{\int_0^{2\pi}\int_0^{\pi}\left|\sigma_{\theta\theta}^{\mathrm{AIM}}-\sigma_{\theta\theta}^{\mathrm{ref}}\right|^2\sin\theta d\theta d\phi}{\int_0^{2\pi}\int_0^{\pi}\left|\sigma_{\theta\theta}^{\mathrm{ref}}\right|^2\sin\theta d\theta d\phi}\right)^{1/2} \tag{2.11}$$

The MOM solution is used as reference for the plates, whereas both analytical Mie series and numerical MOM results are used as reference for the spheres. The AIM parameters are chosen to minimize the iterative solution time, which is often the dominant computational cost, subject to the following error constraints: $err_{\theta\theta}<0.5\%$ with respect

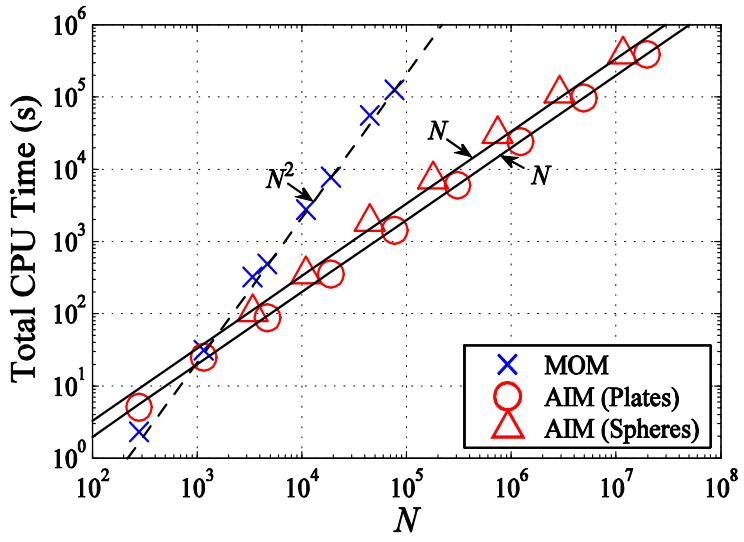
to the reference MOM solution for the plates (when the MOM solution is possible); $err_{\theta\theta} < 1\%$ with respect to both the reference Mie series and MOM results for the spheres (when the MOM solution is possible) (Tables 2.1-2.2).

Table 2.1: Parameters for scattering analysis of plates

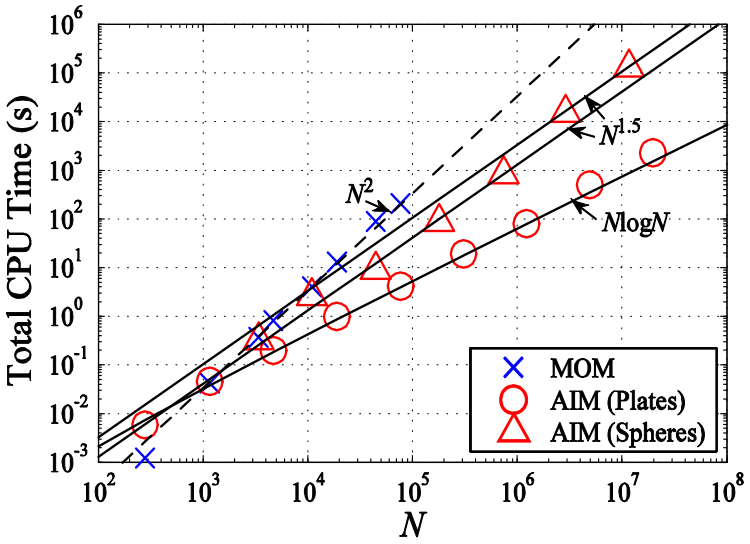
$L_{\mathrm{P}}(\lambda)$	N	γ	N_{C}	Reference	$err_{\theta\theta}(\%)$
1	280	3	$12\times12\times4$	MOM	0.10
2	1 160	3	$20\times20\times4$	MOM	0.36
4	4 720	3	$36\times36\times4$	MOM	0.42
8	19 040	3	$72\times72\times4$	MOM	0.34
16	76 840	3	$144\times144\times4$	MOM	0.46
32	306 560	3	$288\times288\times4$	-	
64	1 227 520	3	$576\times576\times4$	-	
128	4 912 640	3	$1\,152\times1\,152\times4$	-	
256	19 655 680	3	$2\,304\times2\,304\times4$	-	

Table 2.2: Parameters for scattering analysis of spheres

$L_{\mathrm{S}}(\lambda)$	N	γ	N_{C}	Reference	$err_{\theta\theta}(\%)$
1	3 384	3	$24\times24\times24$	MOM	0.19
				Mie	0.93
2	10 947	3	$48\times48\times48$	MOM	0.09
				Mie	0.97
4	44 595	3	$64\times64\times64$	MOM	0.17
				Mie	0.97
8	179 130	3	$128\times128\times128$	-	
				Mie	0.97
16	742 059	3	$256\times256\times256$	-	
				Mie	0.80
32	2 903 916	3	$512\times512\times512$	-	
				Mie	0.97
64	11 601 048	3	$1\,024\times1\,024\times1\,024$	-	
				Mie	0.98

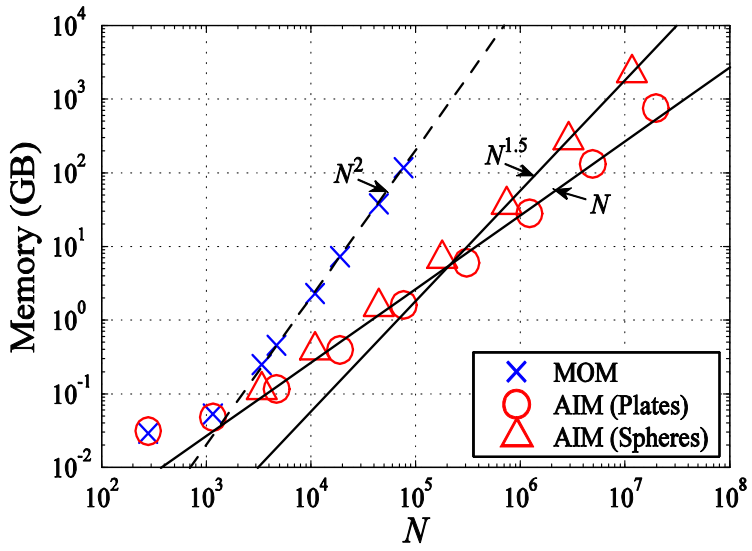


(a)



(b)

Figure 2.3: AIM vs. MOM for PEC plates and spheres as the scatterer sizes are increased. (a) Matrix fill time. (b) Solution time per iteration. (c) Memory requirement.



(c)

Figure 2.3: Continued.

Fig. 2.3 compares the computational requirements of AIM and MOM as observed on the Ranger cluster. These data are obtained from the MPI/OpenMP based parallel implementation of the methods detailed in Chapters 3 and 4; specifically, the timing data are obtained by multiplying the observed wall-clock time with the number of cores used and the memory data are obtained by summing the memory required for all cores. To minimize the effect of parallelization inefficiencies, the simulations in Fig. 2.3 use only the minimum number of cores dictated by the memory requirements and only one core per processor. The results observed in Fig. 2.3 agree well with the theoretically expected computational complexity trends. Notice that AIM accelerated MOM solvers outperform classical solvers in all performance metrics when N is as few as 10^3. Several irregularities are evident in the observed data: (i) In Fig. 2.3(b), a jump in the solution time is observed for the L_s=32λ sphere case. A closer investigation of the data shows that this jump is caused by a disproportionate increase in the time needed for calculating FFTs during the AIM propagation step: The time needed for the $N_{\mathrm{C}} = 512^3$ auxiliary grid (for which 1024 2-D FFTs of size 1024×1024 are calculated) should be ~ 9 larger than that needed for calculating FFTs for an $N_{\mathrm{C}} = 256^3$ grid (for which 512 2-D FFTs

of size 512×512 are calculated) but was observed to be ~17 times larger. For smaller and larger problem sizes, it is observed that the FFT time (as well as the total CPU time) scales as expected; thus, this jump likely indicates the point where the large FFT arrays start to not fit in the faster but limited memory caches. (ii) In Fig. 2.3(c), the memory requirements slightly deviate from the computational complexity line for the largest two plates. This is because of parallelization inefficiencies (specifically, non-parallelized data replication) and is explained in detail in Section 4.4. (iii) In Fig. 2.3(c), the total memory requirement for spheres scales as $O(N)$ when N is relatively small ($N < 10^5$) but as $O(N^{1.5})$ for larger N. This is because the $\mathbf{Z}^{\text{near}}$ matrix initially dominates the memory cost while the $\mathbf{Z}^{\text{FFT}}$ matrix eventually becomes the major memory cost as the size of the spheres increases.

CHAPTER 3: PARALLELIZATION OF THE METHOD OF MOMENTS

This chapter presents (i) MPI parallelization of MOM using a column based decomposition approach, (ii) hybrid MPI/OpenMP parallelization of MOM using a nested column-row based decomposition approach, and (iii) numerical results on the Ranger cluster that compare the parallel performance of the two approaches when applied to the benchmark scatterers introduced in Chapter 2. Note that, the column and nested column-row decomposition based methods can also be formulated as row and nested row-row decomposition based ones, respectively. While this alternative formulation would change the implementation details, e.g., data would be stored using a row-wise instead of a column-wise order in memory, it would not affect the presented results, especially the relative performance of the MPI and hybrid MPI/OpenMP parallelization.

3.1 MPI-MOM: Column Based Decomposition

As detailed in Chapter 2, the dominant computational costs of MOM are the computation of the integrals in (2.2) (matrix fill step), which requires $O\left(N^2\right)$ operations, and the solution of the dense matrix equation $\mathbf{ZI} = \mathbf{V}^{\mathrm{inc}}$ (matrix solve step), which requires $O\left(N^2\right)$ operations per iteration if an iterative solver is used. In a typical distributed-memory MPI implementation, these steps are parallelized by distributing the impedance matrix columns and the current coefficient and incident-field vectors among different MPI processes [31]: If there are P MPI processes, each process is assigned $\sim N / P$ columns of the impedance matrix and the corresponding $\sim N / P$ elements in the current coefficient and incident-field vectors (Fig. 3.1); thus, each process executes $O\left(N^2/P\right)$ operations during the matrix fill step to fill the columns assigned to it and $O\left(N^2/P\right)$ operations per iteration during the matrix solve step to multiply them with the corresponding elements of a trial vector.

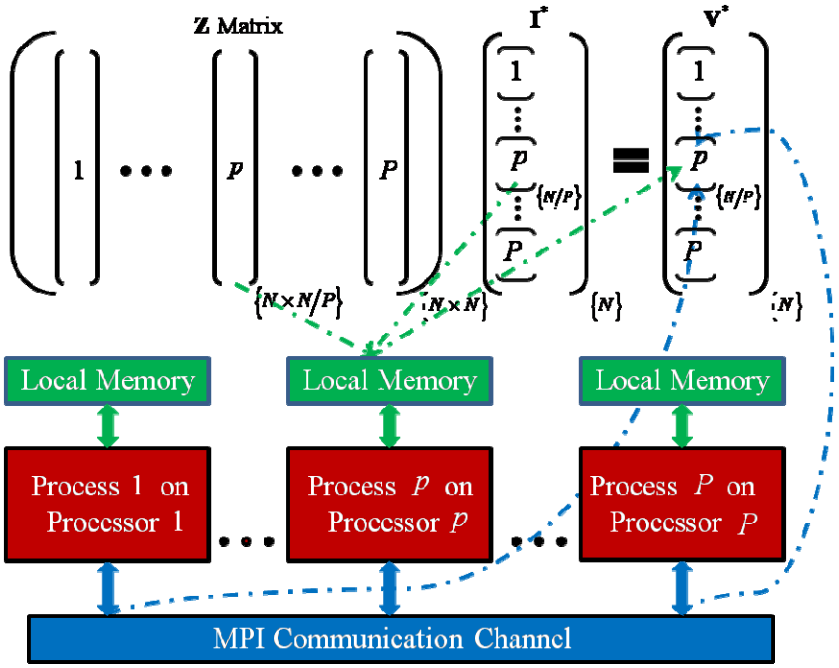
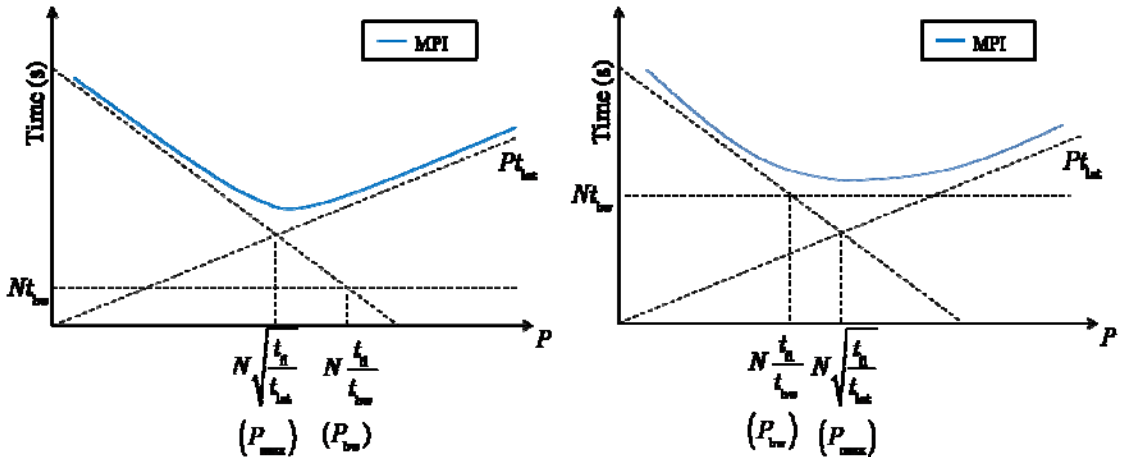


Figure 3.1: MPI parallelization of MOM using column based decomposition on a distributed-memory cluster of single-core processors. The red, green, and blue boxes represent the processors, their local memories, and the common communication channel, respectively. Process p on processor p fills and stores $\sim N/P$ columns of $\mathbf{Z}$ and the corresponding rows of $\mathbf{I}^*$ and $\mathbf{V}^*$ using the local memory. At each iteration, the process calculates its part of the matrix-vector multiplication (green dashed lines), receives contributions to its portion of $\mathbf{V}^*$ from other processes through the channel (blue dashed lines), and sends portions of the result of its matrix-vector multiplication to the other processes through the channel.

Assuming that the geometry and basis/testing function data are replicated, the MPI processes do not communicate during the matrix fill step; thus, this step usually achieves near-ideal parallel scalability. During matrix solve step, however, portions of several vectors (e.g., the trial solution vector $\mathbf{I}^*$ and the residual vector $\mathbf{V}^{\text{inc}} - \mathbf{ZI}^*$) must be communicated among MPI processes in order to calculate matrix-vector multiplications and vector norms (Fig. 3.1). Each process must send to (and receive from) the other $P-1$ processes $O\left(N/P\right)$ bytes of data; thus, the processes exchange a total of $O(P^2)$ messages and communicate a total of $O\left(NP\right)$ bytes per iteration.



(a) (b)

Figure 3.2: Parallel scalability of MPI-MOM matrix solve step: (a) Latency limited ($t_{\text{bw}} < \sqrt{t_{\text{fl}} t_{\text{lat}}}$ and $P_{\max} < P_{\text{bw}}$) (b) Bandwidth limited ($t_{\text{bw}} > \sqrt{t_{\text{fl}} t_{\text{lat}}}$ and $P_{\text{bw}} < P_{\max}$).

To analyze the scalability of the matrix solve step, let t_{lat} denote the latency and t_{bw} the one over the bandwidth of the network; then, $t_{\text{lat}} + Nt_{\text{bw}} / P$ is the time required to communicate N / P bytes of data. If the time required to compute one floating point operation is t_{fl}, then each process requires a total of $O(N^2 t_{\text{fl}} / P + Pt_{\text{lat}} + Nt_{\text{bw}})$ seconds per iteration. Clearly, the parallel scalability of the matrix solve step is limited by the communication time and can be either latency or bandwidth limited depending on the cluster (Fig. 3.2). The time required per iteration can be reduced at best to $O(N[\sqrt{t_{\text{fl}} t_{\text{lat}}} + t_{\text{bw}}])$ seconds using $P_{\max} \sim N\sqrt{t_{\text{fl}} / t_{\text{lat}}}$ processes (using more than $P_{\max}$ processes *increases* the matrix solve time). Note that in the bandwidth limited scenario, parallel performance gains are minimal beyond $P_{\text{bw}} \sim Nt_{\text{fl}} / t_{\text{bw}}$ processes (Fig. 3.2) When the above MPI-MOM is deployed on a cluster of single-core processors, as visually depicted in Fig. 3.1, each active processor is assigned a different MPI process. In contrast, on a cluster of multi-core processors, where each processor consists of multiple computing elements (cores), each active processor should be assigned multiple MPI processes to utilize all the available computational power. Ideally, each core of a processor can be assigned a different MPI process (Fig. 3.3); in practice, however,

performance suffers when all the cores of the processors are used and typically one or more cores are left idle [15, 32]. In addition to the general problem of the limited memory bandwidth available to the multiple cores within the processor, the performance of the MPI-MOM approach is hampered also because it does not account for the memory and communication channel hierarchy of the cluster; specifically, the parallel decomposition algorithm does not account for the shared memory and potentially faster communication channel among the cores of each processor. Thus, compared to an approach that exploits the cluster hierarchy (see next section), the MPI-MOM approach requires additional intra- as well as inter-processor MPI messages; as the number of active cores in a processor increases, there are more messages sent/received using the communication channel of the processor, which increases the impact of latency and limits "multi-core scalability". (It should be noted that "multi-core aware" MPI implementations exist [15, 33-35]; unfortunately, these are currently either in development or are proprietary).

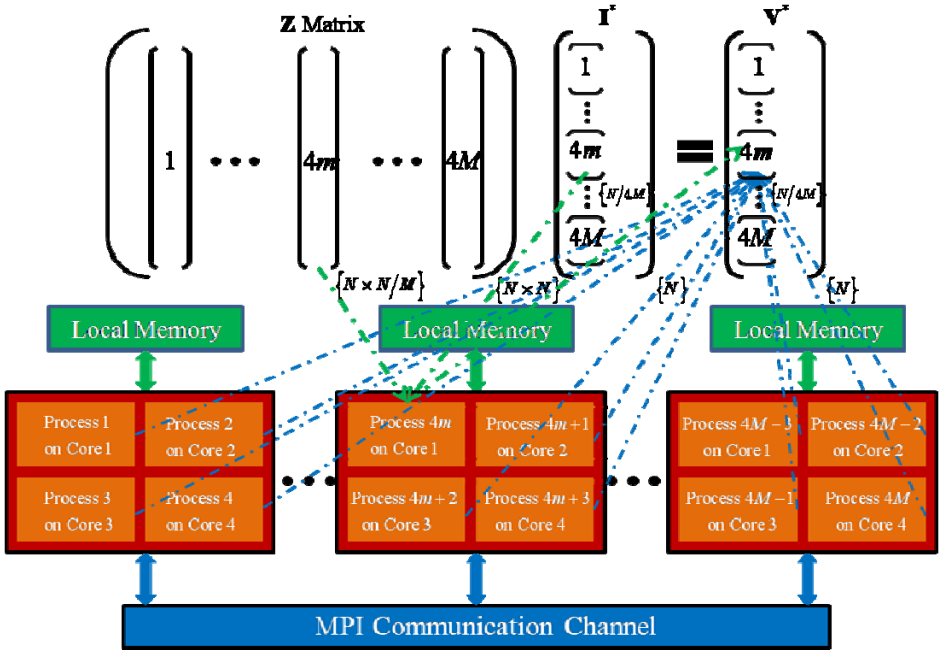
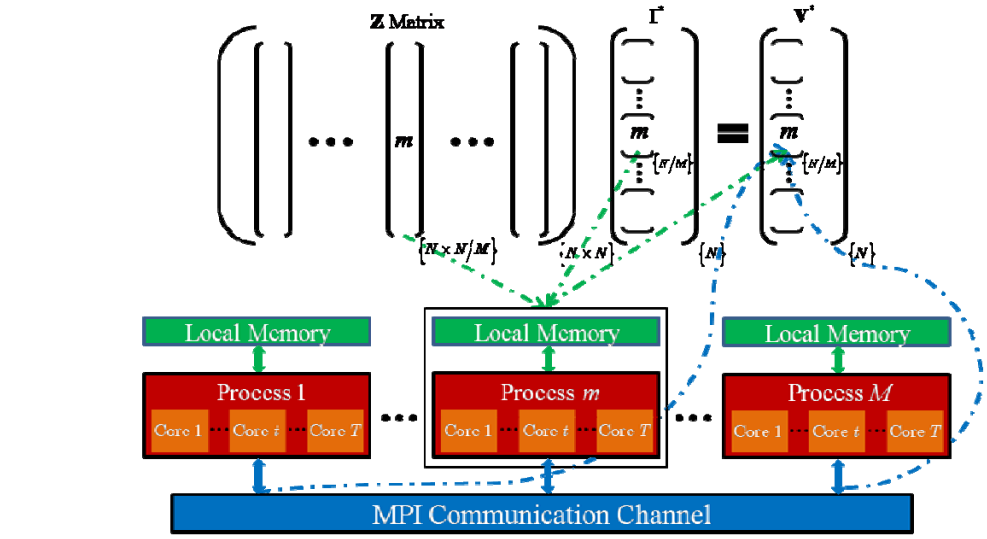


Figure 3.3: MPI parallelization on a cluster of four-core processors when each core is assigned one MPI process. The red, orange, green, and blue boxes represent the processors, their computing cores, the local memories shared by the cores, and the common communication channel, respectively.

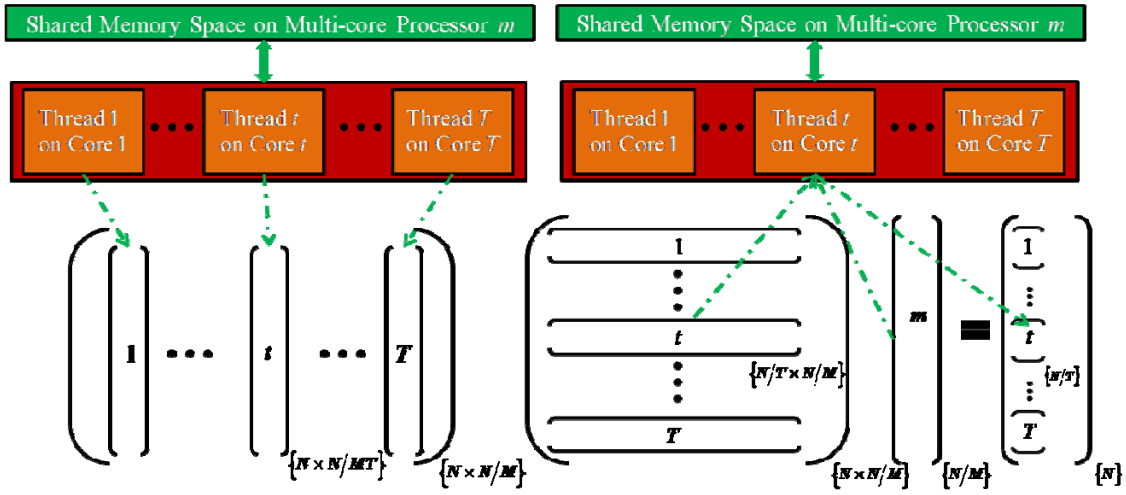
3.2 Hybrid MPI/OpenMP-MOM: Nested Column-Row Decomposition

An alternative approach on clusters of multi-core processors is to use a combination of shared and distributed memory parallelization [16-19]; such a hybrid MPI/OpenMP-MOM scheme can be developed starting from an MPI-MOM implementation. In this approach, each active processor is assigned only one MPI process that initiates multiple OpenMP threads on the processor; typically, one OpenMP thread is initiated per active core, i.e., if M processors and T cores in each processor (a total of MT cores) are active then there are M MPI processes and T OpenMP threads for each process. When the hybrid scheme is executed on a cluster of multi-core processors, it reduces to a pure distributed-memory parallel implementation when one thread per MPI process is used ($M \neq 1,\ T = 1$) and it reduces to a pure shared-memory parallel implementation when one MPI process with multiple threads is used ($M = 1,\ T \neq 1$).

In this approach, the MOM computations are parallelized using a nested decomposition (Fig. 3.4). First, each of the M processes is assigned $\sim N / M$ columns of the impedance matrix and the corresponding $\sim N / M$ elements in the current coefficient and incident-field vectors. Then, each process initiates T threads for computing MOM operations: (i) Matrix fill: Each thread fills $\sim 1 / T$ of the impedance matrix columns assigned to its process; thus, each of the MT threads executes $O\left(N^2 / MT\right)$ operations to fill $\sim N / MT$ columns of the impedance matrix. (ii) Matrix solve: Each thread multiplies $\sim 1 / T$ of the rows in each of the impedance matrix columns assigned to its process; thus, each of the MT threads executes $O\left(N^2 / MT\right)$ operations per iteration to multiply the $\sim N / T$ rows of the impedance matrix with the corresponding trial vector.



(a)



(b) (c)

Figure 3.4: Hybrid MPI/OpenMP parallelization of MOM using nested column-row based decomposition on a cluster of multi-core processors. (a) Workload decomposition among processes: Process m is assigned $\sim N / M$ columns of $\mathbf{Z}$ and the corresponding rows of $\mathbf{I}^*$ and $\mathbf{V}^*$. Each process executes T OpenMP threads. (b) Matrix fill step: Each thread of process m fills $\sim N / MT$ columns of $\mathbf{Z}$ and stores it in the shared memory. (c) Matrix solve step: Each thread of process m multiplies $\sim N / T$ rows of the $\sim N / M$ columns of $\mathbf{Z}$ and writes the result in the shared memory.

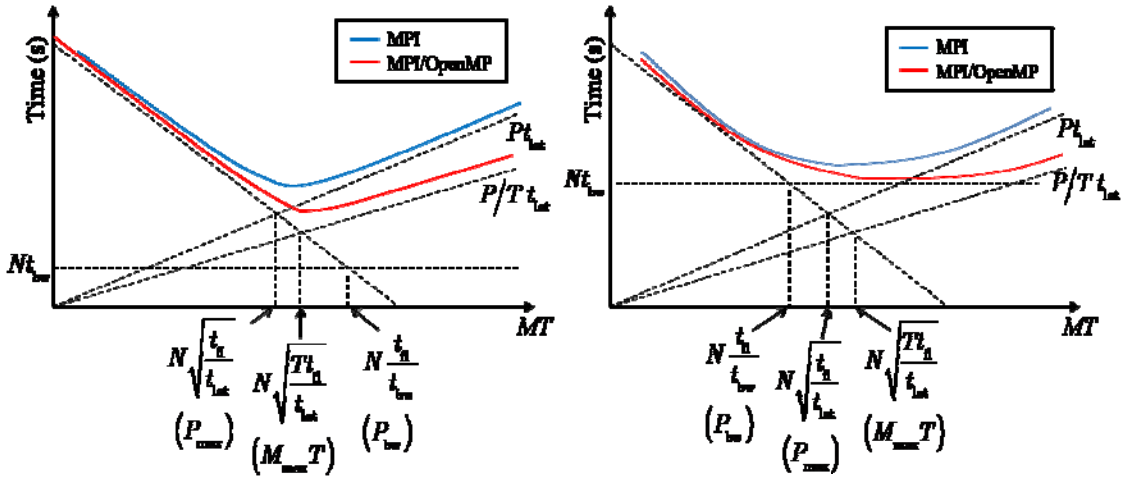
Notice that the impedance matrix is decomposed column and row wise among the threads during the matrix fill and solve steps, respectively. The row-wise division during matrix solve step is natural as it avoids any memory overlap (and thus OpenMP critical

regions) among threads when storing the results of the matrix-vector multiplication in $\mathbf{V}^*$[1]. A similar row-wise division could be used for the matrix fill step; indeed, there is little difference between using a column-wise or a row-wise division when filling the dense MOM impedance matrix. (A column-wise division does maximize contiguous memory access for each thread if the data is stored in column-wise order in memory; however, the resulting buffering and pre-fetching related advantages are minimized by the very large number of operations required for filling each matrix entry). A column-wise division is used when parallelizing the MOM matrix fill step mainly for pedagogical reasons, i.e., to maintain symmetry with the parallelization of AIM, where a column-wise division is more efficient for the matrix fill step (see Chapter 4).

Assuming the geometry and basis/testing function data are replicated over different MPI processes, the matrix fill step of the hybrid MPI/OpenMP implementation exhibits near-ideal parallel scalability because there is neither communication among processes nor memory overlap among threads (although memory bandwidth and cache coherency problems can limit scalability). During the matrix solve step, however, portions of vectors must be either exchanged by inter-processor MPI messages (among the threads that do not share memory) or shared via intra-processor OpenMP directives (among the threads that share memory). To reduce communication costs, only one thread per process (one core per processor) participates in inter-processor communication, i.e., at each iteration, each process sends only one message to (and receives one message from) the other $M-1$ processes and each message contains $O\left(N/M\right)$ bytes of data; thus, the processes exchange a total of $O(M^2)$ messages and communicate $O\left(NM\right)$ bytes of data per iteration. Because each process requires a total of

[1] If column-wise approach is used, then after each thread multiplies the corresponding elements from the $\mathbf{I}^*$ vector with the $\mathbf{Z}$ matrix columns, the whole right hand size vector $\mathbf{V}^*$ is updated with the multiplication results from each thread. This will cause memory overlap as this $\mathbf{V}^*$ vector is shared in the memory among multiple threads.

$O(N^2 t_{\text{fl}} / (MT) + Mt_{\text{lat}} + Nt_{\text{bw}})$ seconds for computation and communication per iteration, the parallel scalability of the hybrid MPI/OpenMP-MOM is limited just like that of the MPI-MOM. Nevertheless, the hybrid approach should exhibit better scalability because it reduces the impact of latency by communicating larger chunks of vectors using fewer messages; this is detailed next.



(a) (b)

Figure 3.5: Parallel scalability of hybrid MPI/OpenMP-MOM vs. MPI-MOM matrix solve step: (a) Latency limited ($P_{\max} < M_{\max} T < P_{\text{bw}} = M_{\text{bw}} T$) (b) Bandwidth limited ($P_{\text{bw}} = M_{\text{bw}} T < P_{\max} < M_{\max} T$).

If T threads are used per process, then the hybrid approach can reduce the time required for the matrix solve step at best to $O(N[\sqrt{t_{\text{fl}} t_{\text{lat}} / T} + t_{\text{bw}}])$ seconds per iteration using $M_{\max} \sim N\sqrt{t_{\text{fl}} / (t_{\text{lat}} T)}$ processes. Note that in the bandwidth limited scenario, performance gains are minimal beyond $M_{\text{bw}} \sim Nt_{\text{fl}} / (t_{\text{bw}} T)$. These costs should be contrasted to those of the pure MPI-MOM using $P = MT$ processes in Section 3.1, e.g., each MPI-MOM process requires a total of $O(N^2 t_{\text{fl}} / (MT) + MTt_{\text{lat}} + Nt_{\text{bw}})$ seconds per iteration that can be reduced at best to $O(N[\sqrt{t_{\text{fl}} t_{\text{lat}}} + t_{\text{bw}}])$ seconds using $P_{\max} \sim N\sqrt{t_{\text{fl}} / t_{\text{lat}}}$ processes. When scalability is latency-limited, the hybrid MPI/OpenMP approach can reduce the minimum time required for the matrix solve step

by a factor of $\sqrt{T}$ compared to the pure MPI approach by using $M_{\max}T$ cores instead of $P_{\max}$ cores ($M_{\max}T = \sqrt{T}P_{\max}$ in this case [Fig. 3.5(a)]). When scalability is bandwidth limited, however, the hybrid approach has little effect on the minimum matrix solve time ($M_{\text{bw}}T \approx P_{\text{bw}}$ in this case [Fig. 3.5(b)]). Nevertheless, the hybrid approach can still be useful because it reduces the *rate of increase* of solution time with the number of cores *beyond* $P_{\max}$ (Fig. 3.5(b)). This means that a smaller time penalty is paid in the matrix solve step when more than $P_{\max}$ cores are employed; these additional cores (i) enable more memory to be accessed[2] and (ii) reduce the time needed for the (near-ideal-scaling) matrix fill step.

3.3 Numerical Results

This section presents numerical results that show the parallel scalability of the MPI-MOM and hybrid MPI/OpenMP-MOM for several of the benchmark spheres and plates described in Chapter 2. The wall-clock time required during the matrix fill step, the average wall-clock time required for one iteration during the matrix solve step, and the memory cost are measured while M, the number of multi-core processors, and T, the number of cores used in each processor, are varied.

First, the differences between scaling M and T are highlighted. Fig. 3.6 shows 3-D log-log-log plots that display the computational requirements for analyzing scattering from the $L_P = 4\lambda$ plate and $L_S = 2\lambda$ sphere along the (logarithmic) *z* axis versus M and T along the (logarithmic) *x* and *y* axes, respectively. These specific examples are simulated because they are large enough to be in the asymptotic complexity regime of MOM and small enough to observe both near-ideal scalability and scalability limitations using relatively small MT. Note that, if an observed computational requirement exhibited ideal parallel scalability with respect to both M and T, then the surface in the 3-D plot

[2] Because the MOM requires $O(N^2)$ memory, the largest problem it can solve is limited by the available memory as well as the simulation time; thus, there is a strong incentive to access more memory.

would be flat and have slope equal to -1 in both coordinate directions at each data point.

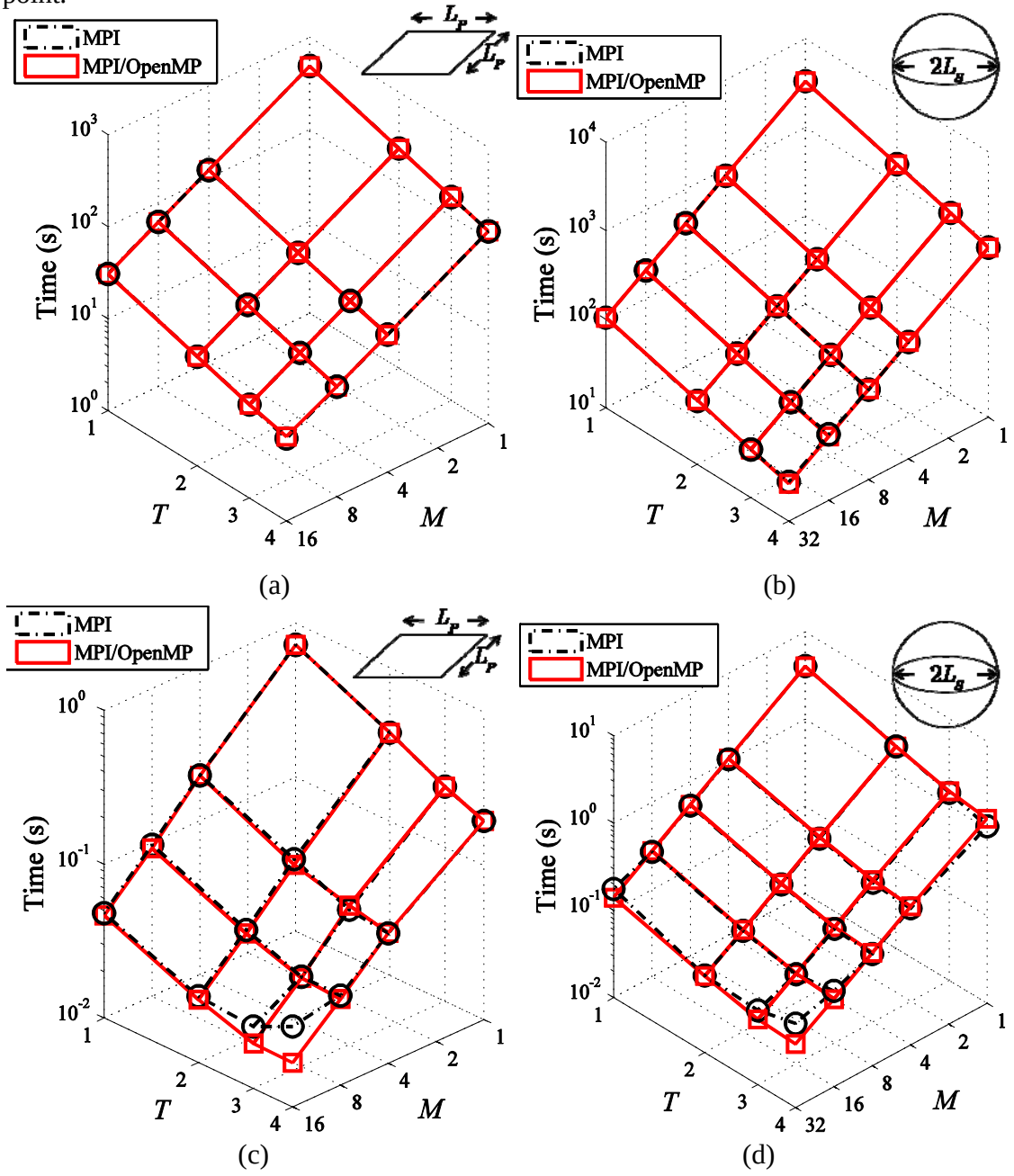
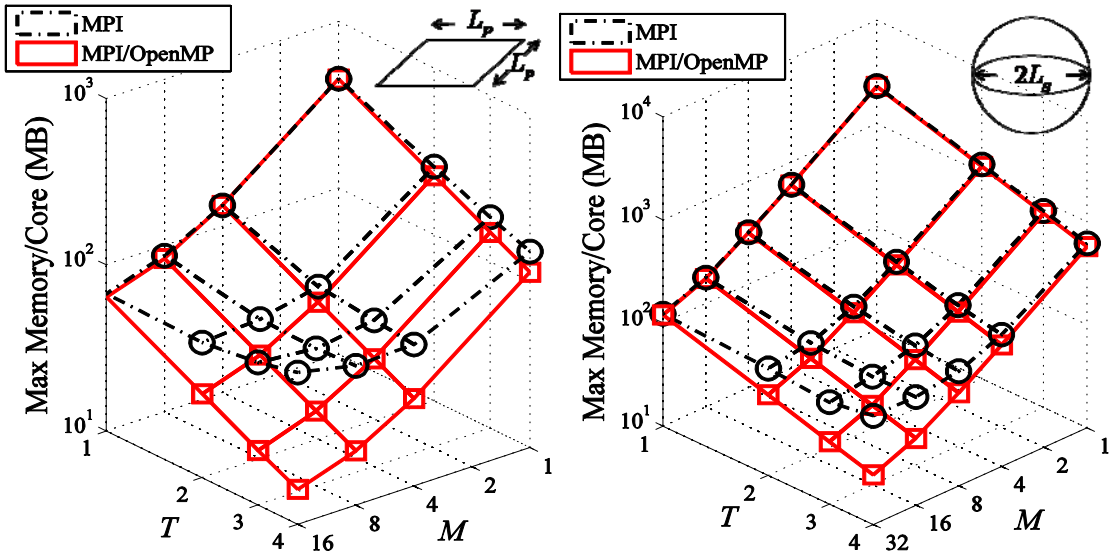


Figure 3.6: Computational requirements for analyzing scattering from an $L_P = 4\lambda$ plate (on the left-hand side) and an $L_S = 2\lambda$ sphere (on the right-hand side) using MPI-MOM and hybrid MPI/OpenMP-MOM. Wall clock time (matrix fill) for (a) plate and (b) sphere. Average wall clock time per iteration (matrix solve) for (c) plate and (d) sphere. Maximum memory needed per core for (e) plate and (f) sphere.



(e) (f)

Figure 3.6: Continued.

Fig. 3.6(a), (c), and (e) on the left-hand side (Fig. 3.6(b), (d), and (f) on the right-hand side) show the wall-clock time during the matrix fill step, the average wall-clock time for one iteration during matrix solve step, and the maximum memory per core[3] during the entire analysis required by the MPI-MOM and hybrid MPI/OpenMP-MOM implementations for the plate (sphere) simulation, respectively. These figures show the following: (i) The matrix fill step of both implementations exhibit near-ideal scalability, which is as expected and indicates that the limited memory bandwidth and cache coherency problems are not critical for this step. (ii) The average wall-clock time per iteration for the MPI-MOM scales well with T when $M = 1$ but it scales poorly with T as M increases. In comparison, the hybrid MPI/OpenMP-MOM exhibits better strong scalability in the matrix solve step, i.e., it can reduce the time per iteration more by using more cores and processors for a fixed problem size. As anticipated, this is because the

[3] "The maximum memory required per core" is found by calculating the maximum of the following data: The peak memory requirement of each MPI process measured during run time for the MPI-MOM implementation; and the peak memory requirement of each MPI process measured during run time divided by the number of OpenMP threads for the hybrid MPI/OpenMP-MOM implementation. This is a convenient measure of memory use on multi-core processors that captures any imbalance among MPI processes but does not account for the imbalance among threads.

communication cost in matrix solve step becomes the dominant cost of the MPI-MOM (as the number of messages is proportional to $O(M^2T^2)$) and the algorithm runs out of scalability earlier than the hybrid MPI/OpenMP implementation that reduces the communication cost by sending larger but fewer messages. (iii) The hybrid MPI/OpenMP-MOM implementation exhibits better strong memory scalability compared to the MPI-MOM as M and T increases; this is because it avoids replicating certain serial data structures among the multiple cores of a processor (see below for details).

Next, more detailed observations are made by using 2-D log-log plots instead of the 3-D plots above. Figs. 3.7 and 3.8 show the computational requirements for several of the benchmark plates and spheres in Chapter 2 along the (logarithmic) *y* axis versus the total number of active cores $P = MT$ along the (logarithmic) *x* axis, respectively.) Because multiple choices of the number of active processors and the number of active cores for each processor can result in the same total number of active cores, the data in the figures are plotted by varying M and fixing T. If an observed computational requirement exhibited ideal parallel scalability with respect to M then the curve in the 2-D plot would be linear and have slope equal to -1 at each data point (i.e., it would be parallel to the ideal scalability tangents). If the requirement exhibited ideal parallel scalability with respect to T then curves with different T in the 2-D plot would coincide.

Fig. 3.7(a), (c), and (e) on the left-hand side (Fig. 3.7(b), (d), and (f) on the right-hand side) show the wall-clock time during the matrix fill step, the average wall-clock time for one iteration during matrix solve step, and the maximum memory among all cores during the entire analysis required by the MPI-MOM (hybrid MPI/OpenMP-MOM) implementation for different plate simulations, respectively. Fig. 3.8(a), (c), and (e) on the left-hand side (Fig. 3.8(b), (d), and (f) on the right-hand side) show the corresponding data for different sphere simulations.

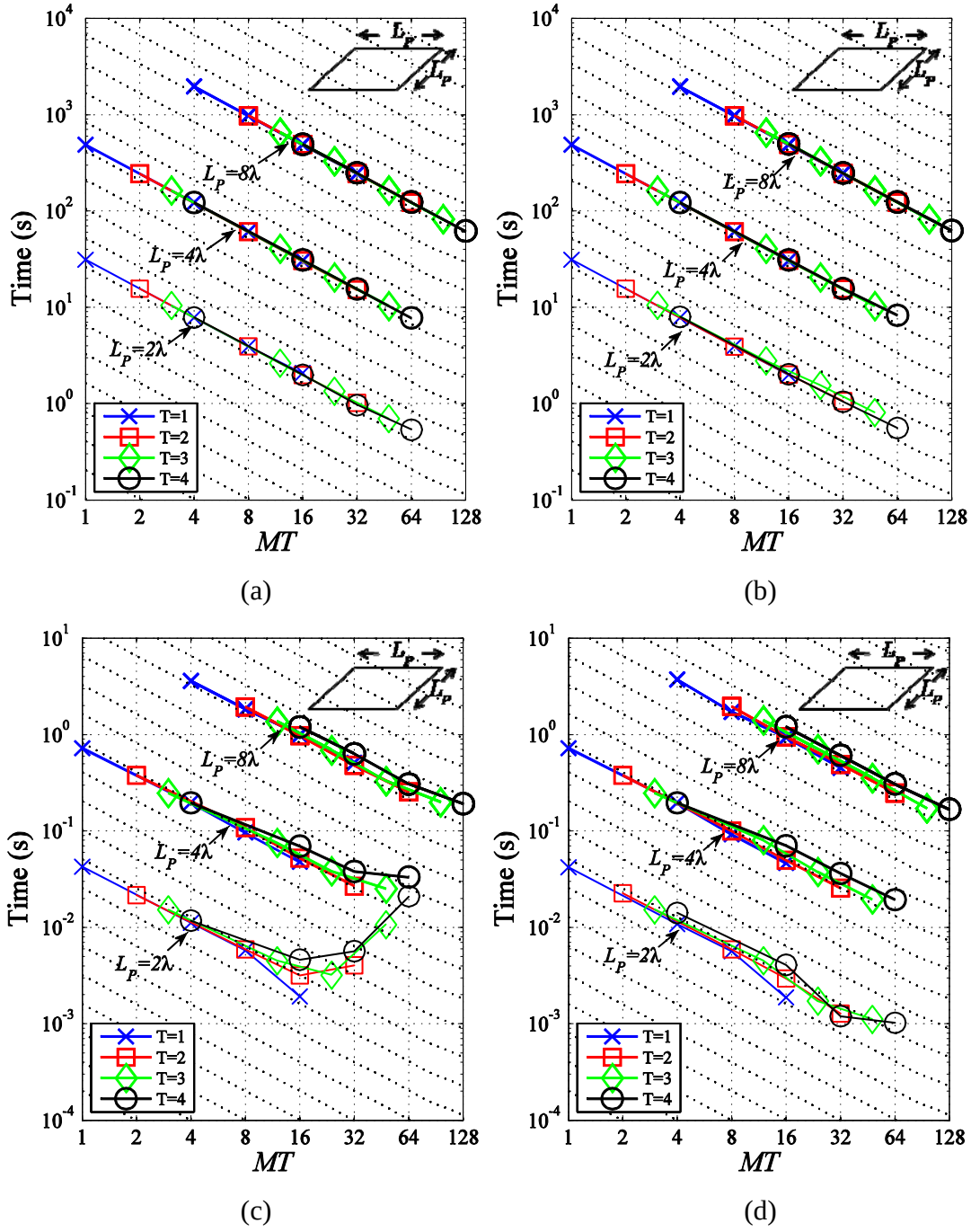
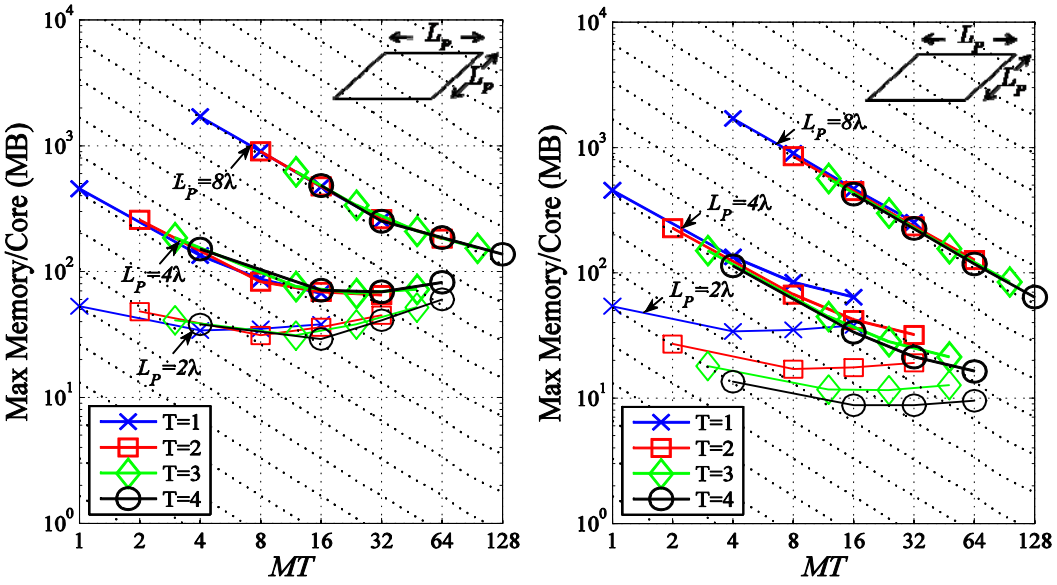


Figure 3.7: Computational requirements for analyzing scattering from $L_P = 2\lambda$, $L_P = 4\lambda$ and $L_P = 8\lambda$ plates using MPI-MOM (on the left-hand side) and hybrid MPI/OpenMP-MOM (on the right-hand side). Wall clock time (matrix fill) for (a) MPI-MOM and (b) hybrid MPI/OpenMP-MOM. Average wall clock time per iteration (matrix solve) for (c) MPI-MOM and (d) hybrid MPI/OpenMP-MOM. Maximum memory needed per core (e) MPI-MOM and (f) hybrid MPI/OpenMP-MOM. Dashed lines are ideal scalability tangents.



(e) (f)

Figure 3.7: Continued.

Figs. 3.7(a)-(b) (Figs. 3.8(a)-(b)) show that the matrix fill step of both implementations exhibit near-ideal scalability for plates (spheres). Moreover, the lines with different color curves that represent different choices of T coincide, i.e., the time required for the matrix fill step does not depend on the particular choice of M and T but is a function of the total number of cores $P = MT$. This implies that the memory and communication hierarchy of Ranger cluster can be ignored for the MOM matrix fill step.

Figs. 3.7(c)-(d) (Fig. 3.8(c)-(d)) show that the matrix solve step of the hybrid MPI/OpenMP-MOM exhibits better scalability and is less sensitive to the choice of T for plates (spheres). It is observed that the fewer number of cores are active in each processor (the smaller T is), the more scalable MPI-MOM is with P; in comparison, the hybrid MPI/OpenMP-MOM scales almost independently of the number of active cores in each processor. Clearly, the memory and communication hierarchy of Ranger cannot be ignored for the MOM matrix solve step and the hybrid MPI/OpenMP-MOM exploits this hierarchy successfully. Moreover, it is evident that the scalabilities of the implementations are latency limited on Ranger.

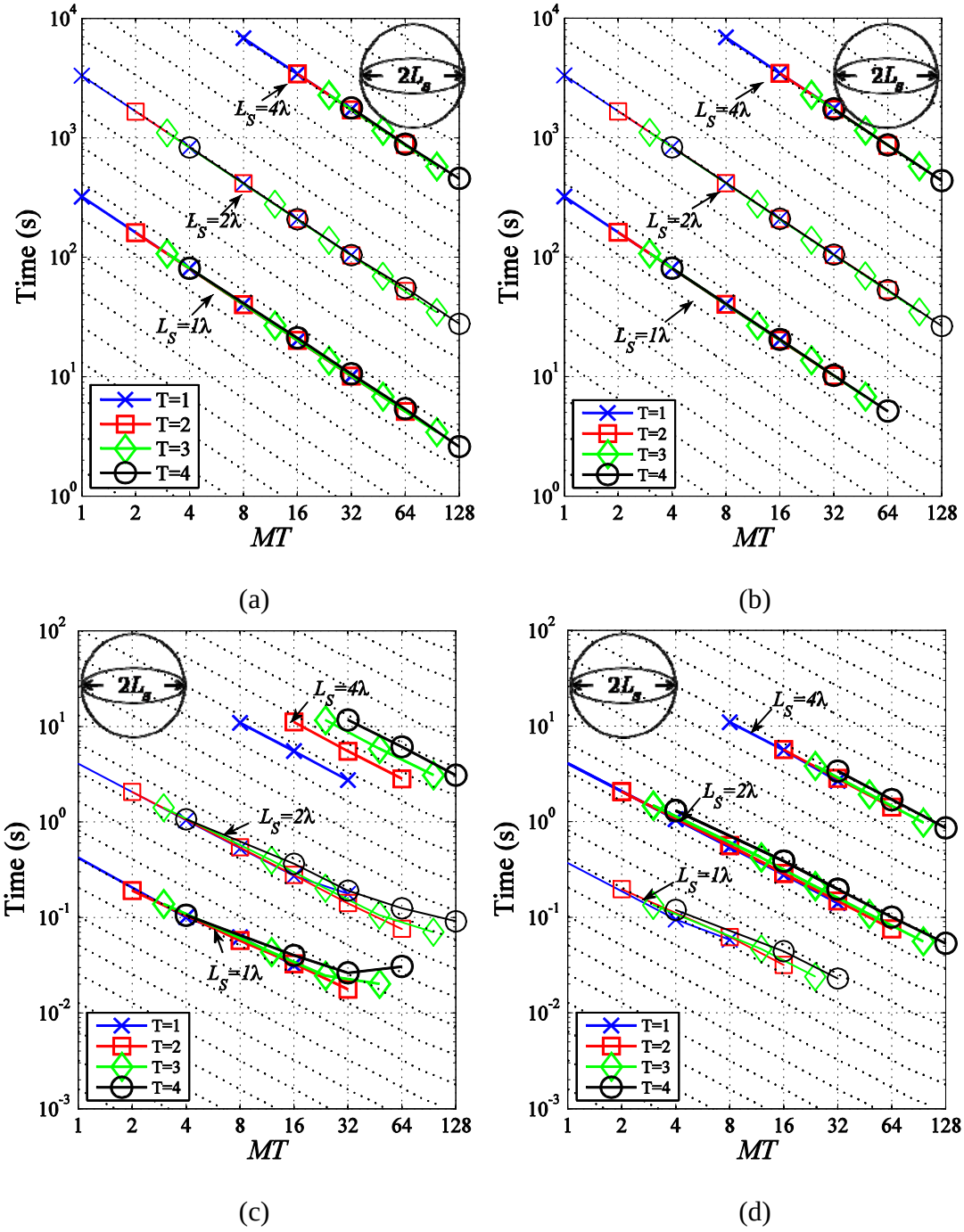
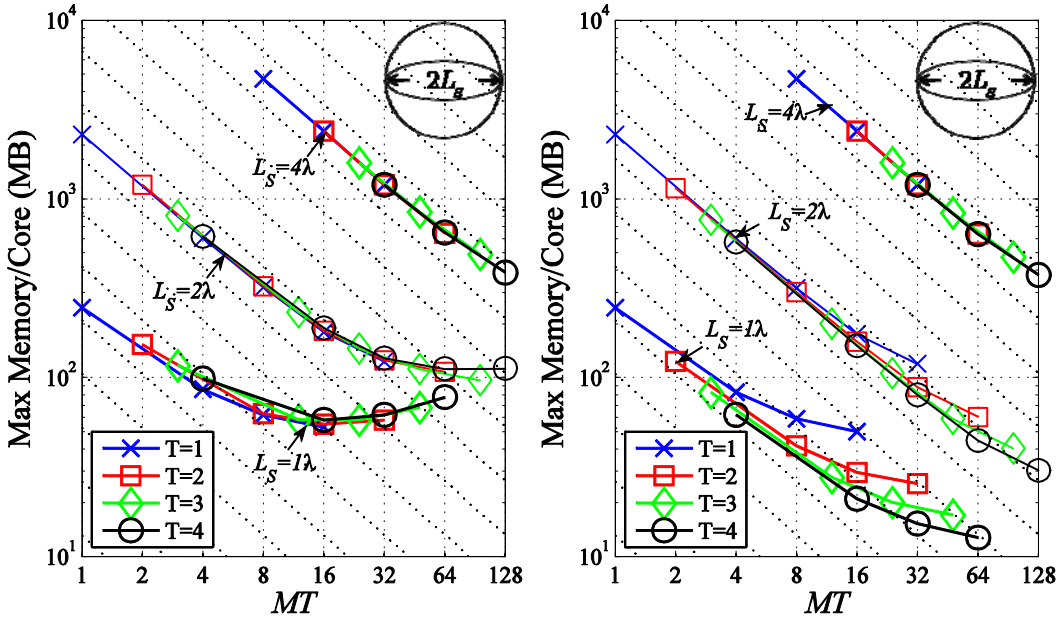


Figure 3.8: Computational requirements for analyzing scattering from $L_S = 1\lambda$, $L_S = 2\lambda$, and $L_S = 4\lambda$ spheres using MPI-MOM (on the left-hand side) and hybrid MPI/OpenMP-MOM (on the right-hand side). Wall clock time (matrix fill) for (a) MPI-MOM and (b) hybrid MPI/OpenMP-MOM. Average wall clock time per iteration (matrix solve) for (c) MPI-MOM and (d) hybrid MPI/OpenMP-MOM. Maximum memory needed per core (e) MPI-MOM and (f) hybrid MPI/OpenMP-MOM. Dashed lines are ideal scalability tangents.



(e) (f)

Figure 3.8: Continued.

Figs. 3.7(e)-(f) (Fig. 3.8(e)-(f)) show that the maximum memory requirement of the hybrid MPI/OpenMP-MOM implementation exhibits better scalability. The memory requirement of both implementations falls down to a constant level as the total number of cores increases; this level is dictated by the operating system, compiler, and parallel library overheads as well as the storage requirements of the non-parallelized data structures, e.g., geometry and basis/testing function data, that are replicated among all MPI processes. Because it only executes one MPI process on each processor and does not replicate the non-parallelized data among different cores of a processor, the MPI/OpenMP-MOM memory requirement is practically independent of T. Note that the memory requirement of the MPI-MOM begins to increase slowly with the number of cores after scaling down to 100 MB for the small plate and sphere examples; this is due to the overhead of auxiliary data structures and operations specifically used for parallelization that scale with the number of cores (but not with the number of unknowns). This overhead is due to both the MPI library and the MPI-MOM

implementation. The memory requirement of the hybrid MPI/OpenMP-MOM will also suffer from this problem as the number of MPI processes increases.

Finally, it should be observed that all computational requirements of both MOM implementations exhibit weak scalability, i.e., the larger the problem size the more cores can be used effectively; in line with the prediction that $P_{\max}$ and $M_{\max}$ are proportional to N.

CHAPTER 4: PARALLELIZATION OF THE ADAPTIVE INTEGRAL METHOD

This chapter presents (i) the MPI parallelization of AIM using a 1-D slab decomposition of the auxiliary grid, (ii) the hybrid MPI/OpenMP parallelization of AIM using a nested slab decomposition of the grid, and (iii) comprehensive numerical results that compare the performance of the two approaches for benchmark scatterers on Ranger.

4.1 MPI-AIM: 1-D Slab Decomposition

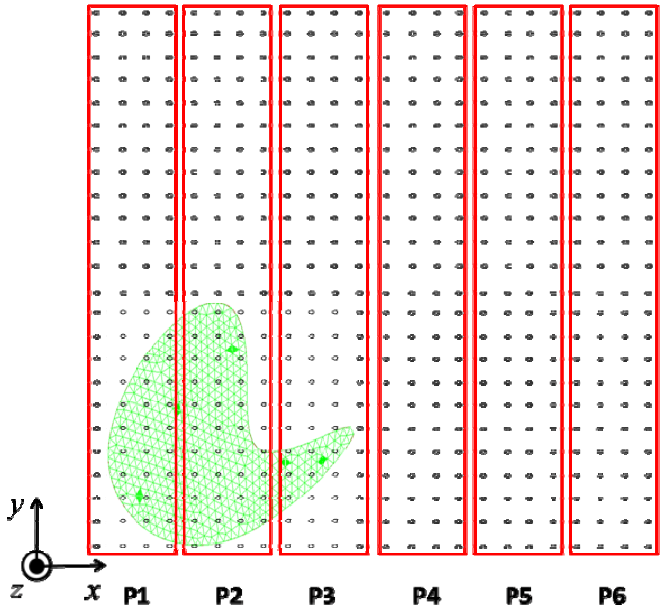


Figure 4.1: 1-D slab decomposition The doubled auxiliary grid is distributed by dividing it along the x dimension among MPI processes. Here and in Figs. 4.2 and 4.3, $P = 6$, $N_{cx} = N_{cy} = 12$, and $N_{cz} = 5$ (z dimension not shown).

The most common method of parallelizing AIM is based on a 1-D slab decomposition of the auxiliary uniform grid [6-9]; in this approach, the grid is partitioned into P slabs along the x dimension and each slab is assigned to one of the P MPI processes (Fig. 4.1). (Note that the AIM grid is doubled because multiplication of a (block) Toeplitz matrix via FFTs [36] requires zero padding of the Green function

matrices and current coefficient vectors). As each MPI process can access only part of the auxiliary grid locally, all AIM steps must be modified.

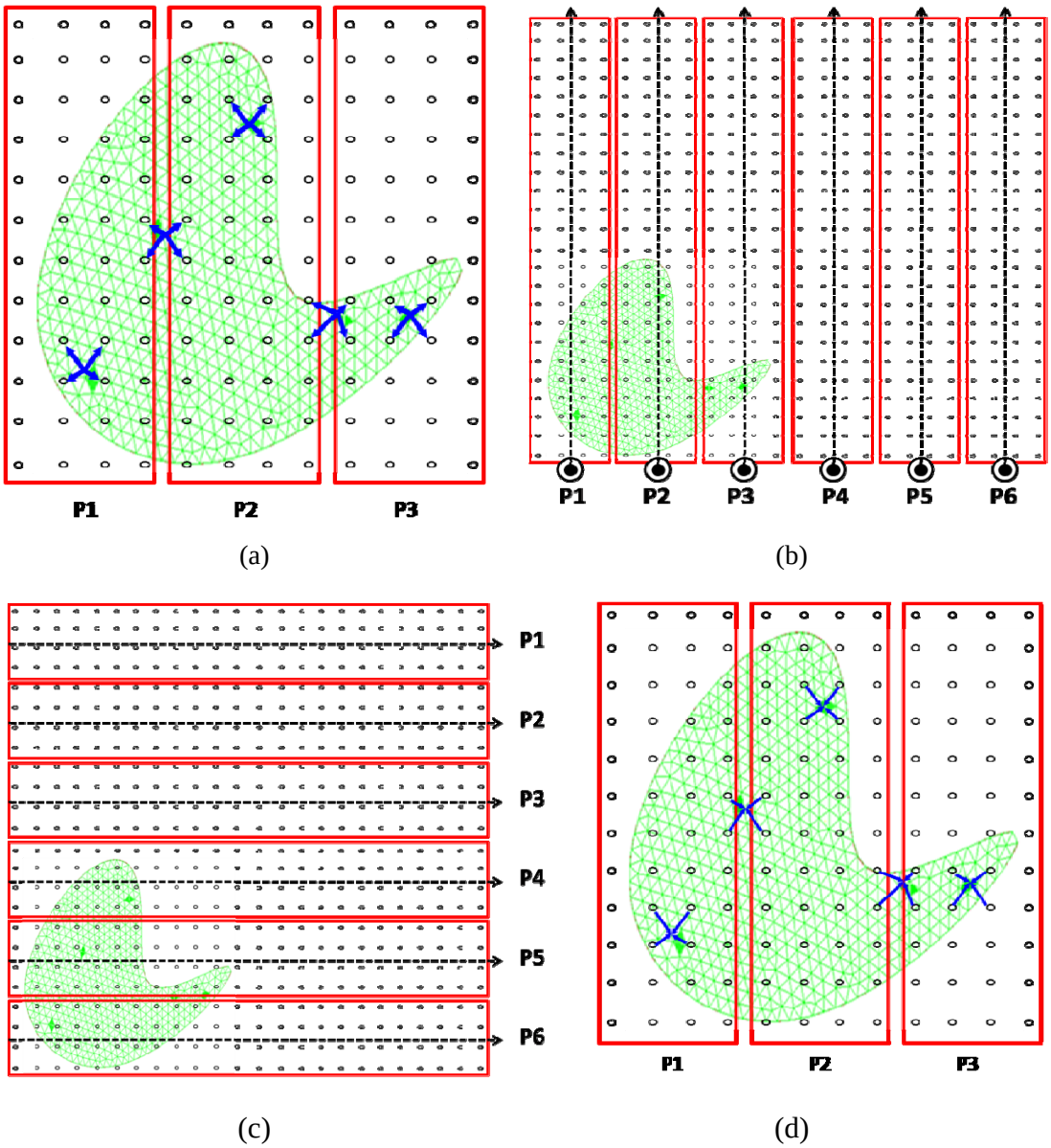


Figure 4.2: Pictorial description of the projection, propagation, and interpolation steps of MPI-AIM. (a) Projection. Only part of the uniform mesh projected by the active processes is shown. (b) Each MPI process computes 4 2-D forward FFTs of size 24×10 in the y and z dimensions followed by a matrix transpose. (c) Each MPI process computes 40 1-D forward FFTs of size 24 along the x dimension. This is followed by point-wise multiplication and 3-D inverse FFTs. (d) Interpolation by the active processes.

Projection Step: Each MPI process projects the current density onto the grid points in its slab of the auxiliary grid (Fig. 4.2(a)). The projection step for basis functions whose associated grid points reside in different slabs is shared among multiple MPI processes (each MPI process computes only the projection onto its slab). Ideally, each process fills and stores $O\left(N/P\right)$ coefficients and executes $O\left(N/P\right)$ operations per iteration for this step, but the primary mesh is non-uniform and only about half of the MPI processes project the current density (due to the doubled grid size) and the workload is unbalanced in this step. Nonetheless, this does not affect the overall performance and scalability of AIM significantly because the projection step typically requires far fewer operations than the propagation and near-zone correction steps.

Propagation Step: First, the 3-D FFTs of currents on the auxiliary grid are computed using the FFTW library [37]. To this end, each MPI process computes 2-D FFTs of (zero-padded) currents at the grid points in its slab along the y and z dimensions (Fig. 4.2(b)); a global matrix transpose redistributes the data among all the processes (or equivalently, the transpose repartitions the grid into slabs along the y dimension); and each MPI process then computes 1-D FFTs along the x dimension (Fig. 4.2(c)). Second, the 3-D FFTs of the currents are multiplied (point-wise) with the 3-D FFTs of the Green function kernel to yield fields (in the spectral domain) at the same grid points. Note that the 3-D FFTs of the Green function kernel is typically pre-computed and is not repeatedly calculated at each iteration. Third, and last, inverse 3-D FFTs of the fields are computed by reversing the steps of the forward 3-D FFTs: Each MPI process computes 1-D inverse FFTs along the x dimension; a second global matrix transpose resumes the original data distribution; and each MPI process computes 2-D inverse FFTs along y and z dimensions. Each process fills and stores $O\left(N_{\mathrm{C}}/P\right)$ bytes of data and executes $O\left(N_{\mathrm{C}}\log N_{\mathrm{C}}/P\right)$ operations per iteration for this step. The workload is perfectly balanced in this step as long as both N_{cx} and N_{cy} are integer multiples of P [8,

9]. *Interpolation Step:* Each MPI process interpolates the fields at the grid points in its slab of the auxiliary grid to the primary surface mesh (Fig. 4.2(d)). Similar to the projection step, the interpolation for testing functions whose associated grid points reside in different slabs is shared among multiple MPI processes (each MPI process computes only the interpolation from its slab). Moreover, each process also fills and stores $O\left(N/P\right)$ bytes and executes $O\left(N/P\right)$ operations per iteration ideally and the load unbalance in the interpolation step mirrors the one in the projection step.

Near-zone Correction Step: The nonzero elements of the sparse near-zone correction matrix $\mathbf{Z}^{\text{near}}$ are distributed by a column based decomposition approach similar to the approach for MPI-MOM in Section 3.1[4], i.e., the (sparse) columns of the $\mathbf{Z}^{\text{near}}$ matrix are distributed among the MPI processes such that each process fills and stores $O(N_{\text{near}} / P)$ matrix entries and executes $O(N_{\text{near}} / P)$ operations per iteration. While the computations can be relatively easily balanced across processes, the distribution of the columns in this step also affects the cost of communication for calculating vector norms during the iterative solution. While there are several general-purpose methods for load-balancing sparse-matrix vector multiplications [38], a method specific to AIM is employed here: The columns are distributed to the processes based on the location of the basis functions by utilizing the auxiliary grid and the slab decomposition.

Just like those for MPI-MOM, the processes for MPI-AIM only communicate during the matrix solve step: (i) During the propagation step, each MPI process must send to (and receive from) the other $P-1$ processes $O\left(N_{\text{C}}/P^2\right)$ bytes of data to transpose

[4] Unlike the case for MOM, where the column and row based decomposition are equivalent, the column based decomposition is more efficient for AIM especially during the matrix fill step. This is because the $\mathbf{Z}^{\text{FFT}}$ contribution to the $\mathbf{Z}^{\text{near}}$ matrix can be found more efficiently one-column-at-a-time than one-row-at-a-time by amortizing the projection and propagation steps: Only one projection and one propagation step (and many interpolation steps) are needed to fill each column of the $\mathbf{Z}^{\text{near}}$ matrix but many projection, propagation, and interpolation steps are needed to fill each row.

matrices for 3-D FFT calculations. (ii) During the near-zone correction step, depending on the distribution of the columns among the processes, each process will send to (and receive from) an average of P_{near} other processes $O(N / P)$ bytes of data[5]; thus, the processes exchange a total of $O(P^2 + PP_{\text{near}})$ messages and communicate a total of $O\left(N_{\text{C}} + NP_{\text{near}}\right)$ bytes of data per iteration.

To analyze the parallel scalability of the above MPI-AIM, let t_{lat} denote the latency, t_{bw} the one over the bandwidth of the network, and t_{fl} the time required to compute one floating point operation as in Chapter 3. Then, each process requires a total of $O([N_{\text{C}} \log N_{\text{C}} + N_{\text{near}}]t_{\text{fl}} / P + [P + P_{\text{near}}]t_{\text{lat}} + [N_{\text{C}} + NP_{\text{near}}]t_{\text{bw}} / P)$ seconds per iteration for the propagation and near-zone correction steps---the two computationally dominant steps of AIM . Because $P \gg P_{\text{near}}$ typically, the parallel scalability of MPI-AIM is limited principally by latency and the time required per iteration can be reduced at best to $O(\sqrt{[N_{\text{C}} \log N_{\text{C}} + N_{\text{near}}]t_{\text{fl}}t_{\text{lat}} + [N_{\text{C}} + NP_{\text{near}}]t_{\text{bw}}t_{\text{lat}}})$ seconds using $P_{\max} \sim \sqrt{[N_{\text{C}} \log N_{\text{C}} + N_{\text{near}}]t_{\text{fl}} / t_{\text{lat}} + [N_{\text{C}} + NP_{\text{near}}]t_{\text{bw}} / t_{\text{lat}}}$ processes. A comparison with MPI-MOM, which can achieve a minimum time per iteration of $O(N[\sqrt{t_{\text{fl}}t_{\text{lat}}} + t_{\text{bw}}])$ using $P_{\max}^{\text{MOM}} \sim N\sqrt{t_{\text{fl}} / t_{\text{lat}}}$ processes (Section 3.1), is in order: Clearly, MPI-AIM is scalable to a significantly fewer number of processes for a given scatterer; e.g., for the benchmark spheres, $P_{\max} \sim \sqrt{[N^{3/2} \log N]t_{\text{fl}} / t_{\text{lat}} + N^{3/2}t_{\text{bw}} / t_{\text{lat}}}$ and for the benchmark plates, $P_{\max} \sim \sqrt{[N \log N]t_{\text{fl}} / t_{\text{lat}} + Nt_{\text{bw}} / t_{\text{lat}}}$. Nonetheless, the minimum time per iteration that MPI-AIM can achieve is significantly smaller because of its reduced complexity, e.g., MPI-AIM requires down to $O(N^{3/4}\sqrt{\log N})$ and $O(N^{1/2}\sqrt{\log N})$

[5] Each process sends at worst to $P_{\text{near}} \sim P$ processes and at best to $P_{\text{near}} \sim 1$ processes. Worst case: The entries of the (sparse) columns assigned to the process are distributed such that multiplication with the corresponding entries of the current-coefficient vector results in $O\left(N\right)$ non-zero entries. Best case: The sparse-matrix vector multiplication assigned to the process results in $O\left(N / P\right)$ non-zero entries, most of which are assigned to the same process. In other words, the union of the near-zone regions of the basis functions assigned to the process covers the entire scatterer in the worst case and covers a small neighborhood of the region assigned to that process in the best case.

seconds (instead of down to $O(N)$ seconds for MPI-MOM) per iteration for the benchmark spheres and plates, respectively.

4.2 Blocking vs. Non-blocking 3-D FFTs

In the above parallelization approach, the matrix transpose at the propagation step can be implemented by using either a (blocking) collective communication or a (non-blocking) point-to-point communication approach. These are detailed and contrasted for MPI-AIM next. To simplify the presentation, it is assumed that N_{cx} and N_{cy} are integer multiples of P.

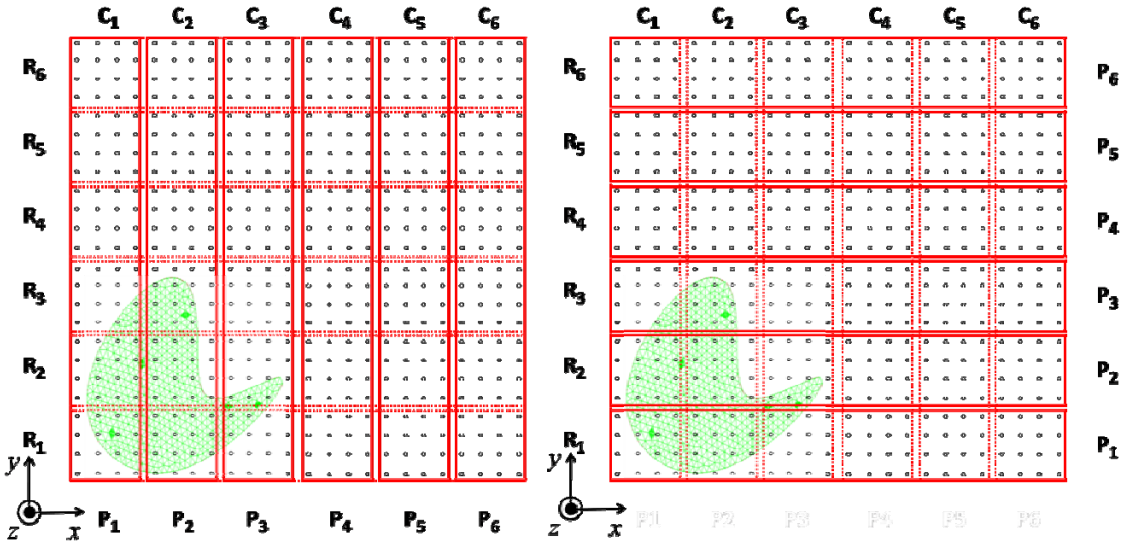


Figure 4.3: Pictorial description of the matrix transpose at the propagation step. Slab decomposition before and after the transpose for the 3-D (a) FFTs and (b) inverse FFTs. The straight and dashed red lines show the slab assigned to each process before and after the transpose, respectively. The symbols $\mathrm{C}_1 - \mathrm{C}_6$ and $\mathrm{R}_1 - \mathrm{R}_6$ identify column slabs and row slabs, respectively.

Consider the matrix transpose during the 3-D FFTs at the propagation step (Fig. 4.3(a)): Each MPI processes is assigned $2N_{cx}/P$ columns (a column slab) of the auxiliary grid before the matrix transpose and $2N_{cy}/P$ rows (a row slab) of the auxiliary grid after the matrix transpose (the number of grid points in the slab assigned to each process is $\left(4N_{cy}N_{cz}\right)2N_{cx}/P$ before and $\left(4N_{cx}N_{cz}\right)2N_{cy}/P$ after the transpose). Thus,

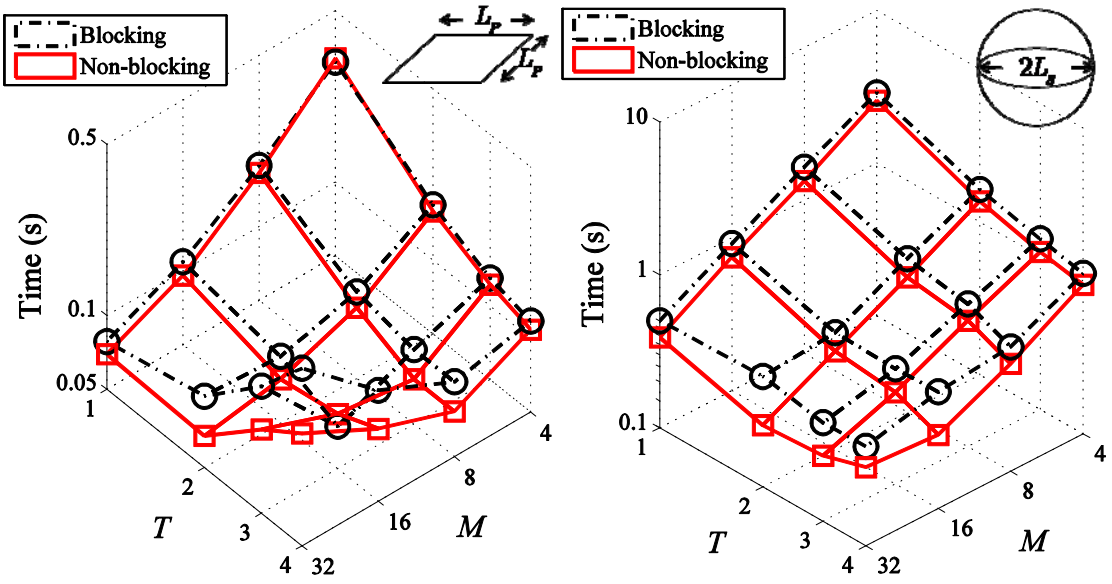
each matrix transpose requires an all-to-all communication where each process receives and sends $O(N_{\mathrm{C}} / P)$ bytes of data.

In the collective communication approach, the processes communicate only after all of them finish computing the 2-D FFTs along the *y*-*z* dimensions in their column slabs and no process can compute the 1-D FFTs along the *x* dimension in their row slabs until all of them have finished sending (and receiving) data. This approach relies on the "MPI_Alltoall" collective communication directive and is very common, e.g., the multi-dimensional FFT subroutines in the FFTW library adopt this approach [37].

In the point-to-point communication approach, first each process issues $P-1$ non-blocking "MPI_IRecv" directives to collect the row slab data. The row slob data is received in small blocks: Let $\mathrm{R}_i\mathrm{C}_j$ denote the block of grid points at the intersection of the i^{th} row and j^{th} column slab; the block contains $2N_{cz}\left(4N_{cx}N_{cy} / P^2\right)$ grid points; and the process P_i receives the blocks $\mathrm{R}_i\mathrm{C}_1 - R_i\mathrm{C}_{j\text{-}1}$ and $\mathrm{R}_i\mathrm{C}_{j+1} - R_i\mathrm{C}_P$ from the corresponding processes. Second, each process computes the 2-D FFTs in its column slab and organizes the columns into small blocks. Third, each process issues $P-1$ non-blocking "MPI_ISend" directives to distribute the column slab data, i.e., process P_j sends the blocks $\mathrm{R}_1\mathrm{C}_j - R_{j-1}\mathrm{C}_j$ and $\mathrm{R}_{j+1}\mathrm{C}_j - R_P\mathrm{C}_j$ to the corresponding processes. Fourth and finally, each process waits for the receive directives to be completed using "MPI_Wait" and then computes the 1-D FFTs in its row slab. Notice that computation and communication are overlapped in this approach. Furthermore, the approach can be specialized to the FFTs in AIM. Specifically, because of the doubled auxiliary grid, only about half of the processes must compute the 2-D forward and 1-D inverse FFTs[6]; importantly, this implies that the number of messages and the total data size

[6] For the forward (inverse) FFTs: The output (input) of the AIM projection (interpolation) step is non-zero current (field) values only at the grid points immediately surrounding the scatterer; thus, only the processes that are assigned the column slabs that contain the scatterer (e.g., $P_1 - P_3$ in Fig. 4.3) must compute 2-D forward FFTs (1-D inverse FFTs), as the remaining processes (e.g., $P_4 - P_6$ in Fig. 4.3) can avoid the FFT computations because the results are zero (are not interpolated back onto the scatterer mesh).

communicated can be reduced by ~2 as approximately half the processors must send data after the forward FFTs and receive data before the inverse FFTs.



(a) (b)

Figure 4.4: 3-D FFTs using collective vs. point-to-point communication. Average wall clock time per iteration required during matrix solve step for (a) the $L_P = 8\lambda$ plate and (b) $L_S = 4\lambda$ sphere.

Fig. 4.4 compares the performance of collective and point-to-point communication approaches for the matrix transpose at the AIM propagation step. The figure shows the average wall clock time per iteration required during matrix solve step of MPI-AIM for a benchmark plate and sphere. The non-blocking FFT implementation shows slightly better scalability and is adopted henceforth.

4.3 Hybrid MPI/OpenMP-AIM: Nested Slab Decomposition

Although the above MPI-AIM is well suited for distributed memory architectures, it is not multi-core scalable for reasons similar to those for MPI-MOM detailed in Section 3.1. To exploit the memory and communication channel hierarchy of multi-core clusters, a hybrid MPI/OpenMP parallelization of AIM is proposed next. As in Chapter 3, it is assumed that each active processor is assigned only one MPI process that initiates

multiple OpenMP threads on the processor (one OpenMP thread per core), i.e., if M processors and T cores in each processor (a total of MT cores) are active then there are M MPI processes and T OpenMP threads for each process.

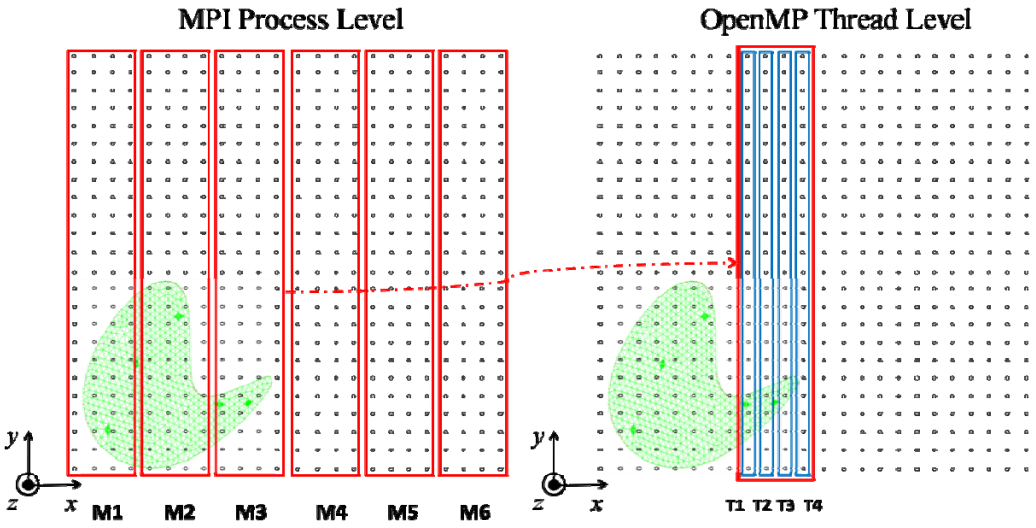
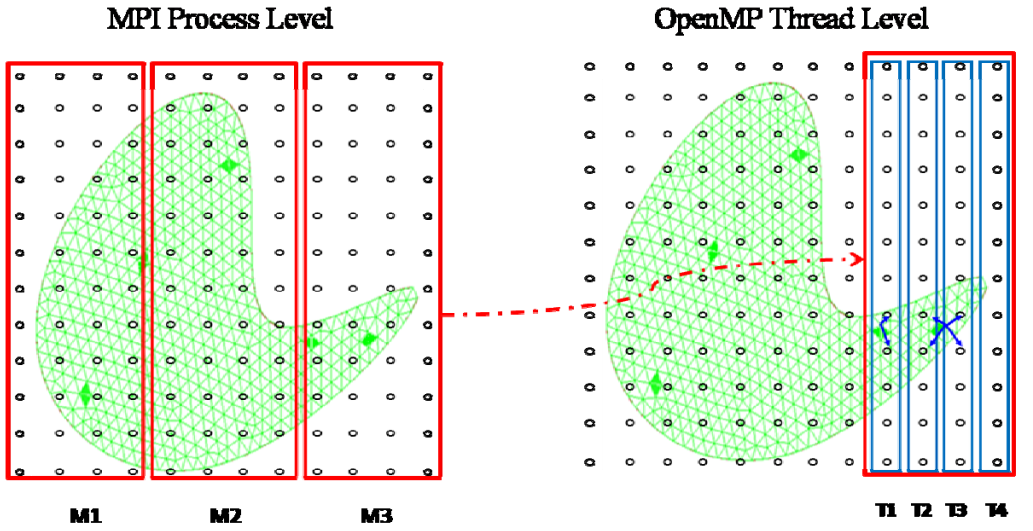


Figure 4.5: Nested 1-D slab decomposition. The doubled auxiliary grid is distributed by dividing it along the x dimension first among MPI processes and then among OpenMP threads. Here and in Fig. 4.6, $M = 6, T = 4$, $N_{cx} = N_{cy} = 12$, and $N_{cz} = 5$ (z dimension not shown).

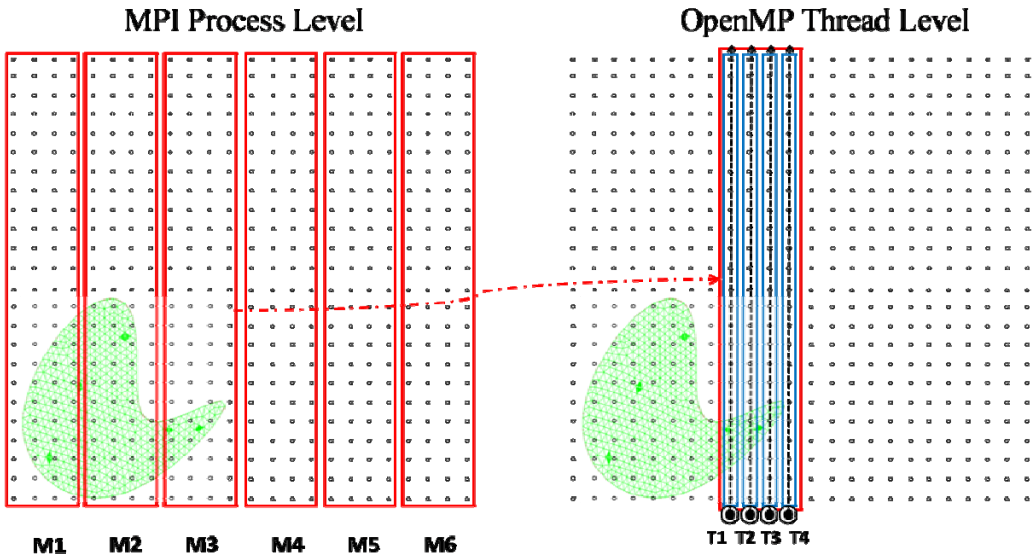
The hybrid MPI/OpenMP-AIM is parallelized using a nested decomposition (Fig. 4.5). In this approach, the grid is partitioned into M slabs along the x dimension and each slab is assigned to one of the M MPI processes (Fig. 4.5). Then, each slab is further partitioned into T sub-slabs each of which is assigned to one of the T OpenMP threads initiated by the process. Because multiple threads share the same memory space, the MPI-AIM steps must be modified accordingly.

Projection Step: Each OpenMP thread projects the current density on the primary mesh onto its sub-slab of the auxiliary grid (Fig. 4.6(a)). The projection step for basis functions whose associated grid points reside in different sub-slabs is shared among multiple OpenMP threads (each OpenMP thread computes only the projection onto its own sub-slab). Ideally, each thread fills $O\left(N/MT\right)$ coefficients and executes $O\left(N/MT\right)$ operations per iteration but the workload among threads is also unbalanced

in this step because of the non-uniform primary mesh (see Section 4.1). Note that there is no memory overlap among threads (and thus no OpenMP critical regions) in this step.

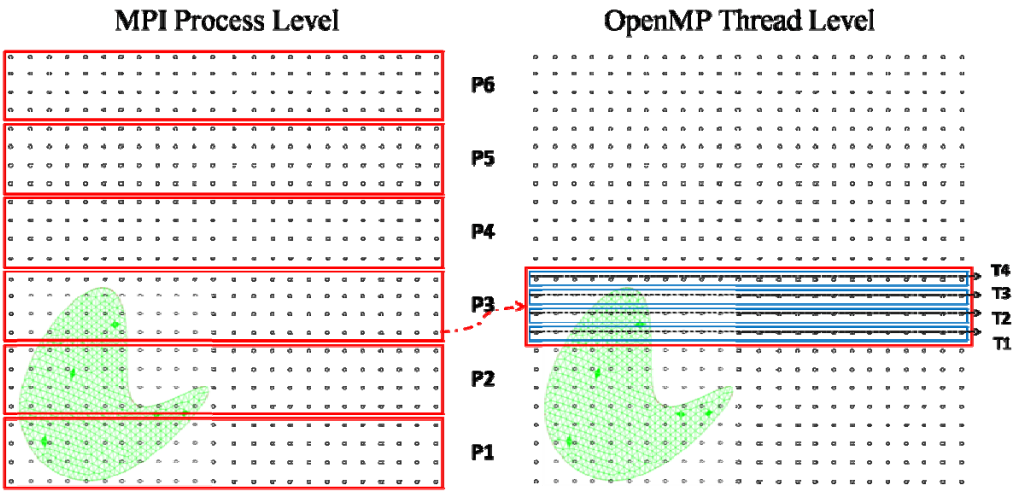


(a)

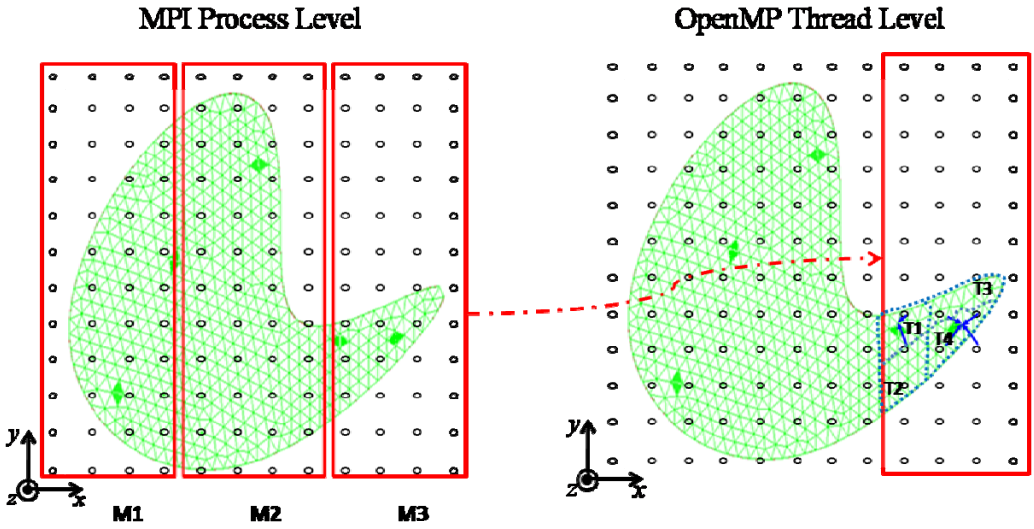


(b)

Figure 4.6: Pictorial description of the projection, propagation, and interpolation steps of hybrid MPI/OpenMP-AIM. (a) Projection. Only part of the uniform mesh projected by the active threads is shown. (b) Each thread computes 10 forward FFTs of size 24 in the y dimension and 24 forward FFTs of size 10 in the z dimension followed by a matrix transpose. (c) Each thread computes 10 forward FFTs of size 24 along x dimension, point-wise multiplication within its sub-slab, and 10 inverse FFTs of size 24 along x dimension. (d) Interpolation by the active threads.



(c)



(d)

Figure 4.6: Continued.

Propagation Step: First, the 3-D FFTs of the current on the auxiliary grid are computed: Each thread computes the 2-D FFTs in y and z dimensions of (zero-padded) currents on its sub-slab of the auxiliary grid (Fig. 4.6(b)); a global matrix transpose at the MPI process level repartitions the grid into slabs in y dimension; these slabs are divided into T sub-slabs and each OpenMP thread computes 1-D FFTs along the x dimension in its sub-slab (Fig. 4.6(c)). Second, the 3-D FFTs of the currents are multiplied (point-wise) with the (pre-computed) 3-D FFTs of the Green's function kernel: Each OpenMP thread

multiplies the Green's function kernel in its sub-slab of the grid. Third, and finally, 3-D inverse FFTs are computed by reversing the above steps. Each thread fills $O\left(N_{\mathrm{C}}/MT\right)$ bytes of data and executes $O\left(N_{\mathrm{C}}\log N_{\mathrm{C}}/MT\right)$ operations per iteration for this step. The workload is perfectly balanced in this step as long as both N_{cx} and N_{cy} are integer multiples of MT.

Interpolation Step: The threads interpolate the fields at the grid points onto the testing function on the primary surface mesh. The workload is parallelized as follows: First, the 1-D slab decomposition is used to assign testing functions to each MPI process. Then, the testing functions (and not the grid points) assigned to each process are distributed among its OpenMP threads and each thread interpolates the fields from the appropriate grid points to the testing functions assigned to it (Fig. 4.6(d)). This unknown-based decomposition approach avoids memory-write overlap among threads and OpenMP critical regions. Ideally, each thread fills $O\left(N/MT\right)$ coefficients and executes $O\left(N/MT\right)$ operations per iteration; while the workload of threads in each process is well balanced, the workload across processes is unbalanced mirroring the projection step.

Near-zone Correction Step: The near-zone correction operations are distributed among threads using a nested decomposition approach similar to the hybrid MPI/OpenMP-MOM: First, the columns of the $\mathbf{Z}^{\mathrm{near}}$ matrix are distributed among the MPI processes such that each process is assigned $\sim N_{\mathrm{near}}/M$ matrix entries and an arbitrary number of columns[7]. Then, each process initiates T threads during matrix fill and solve steps. (i) Matrix fill: Each thread fills $\sim 1/T$ of the columns assigned to its process; thus, each of the MT threads executes $O\left(N_{\mathrm{near}}/MT\right)$ operations. (ii) Matrix solve: Each thread multiplies $\sim 1/T$ of the rows in each of the columns assigned to its

[7] The number of columns assigned to each process depends on the sparsity pattern of the $\mathbf{Z}^{\mathrm{near}}$ matrix, which reflects the distribution of unknowns on the scatterer mesh.

process; thus, each of the MT threads executes $O\left(N_{\text{near}}/MT\right)$ operations per iteration to multiply the $\mathbf{Z}^{\text{near}}$ matrix with the corresponding trial vector.

To reduce communication costs during the matrix solve step, only one thread per process (one core per processor) in MPI/OpenMP-AIM participates in inter-processor communication: (i) During the propagation step, each MPI process must send to (and receive from) the other $M-1$ processes $O\left(N_{\text{C}}/M^2\right)$ bytes of data to transpose matrices for 3-D FFT calculations. (ii) During the near-zone correction step, depending on the distribution of the columns among the processes, each process will send to (and receive from) an average of M_{near} other processes $O(N/M)$ bytes of data; thus, the processes exchange a total of $O(M^2+MM_{\text{near}})$ messages and communicate a total of $O\left(N_{\text{C}}+NM_{\text{near}}\right)$ bytes of data per iteration. Because a total of $O([N_{\text{C}}\log N_{\text{C}}+N_{\text{near}}]t_{\text{fl}}/\left(MT\right)+[M+M_{\text{near}}]t_{\text{lat}}+[N_{\text{C}}+NM_{\text{near}}]t_{\text{bw}}/M)$ seconds per iteration is required for the propagation and near-zone correction steps, the parallel scalability of the hybrid MPI/OpenMP-AIM is limited by latency just like that of the MPI-AIM. This implies that the hybrid approach will exhibit better scalability because it reduces the impact of latency by communicating larger chunks of vectors using fewer messages. Specifically, assuming that $M \gg M_{\text{near}}$ (as is typical), the hybrid approach can reduce the time required for the matrix solve step at best to $O(\sqrt{\left([N_{\text{C}}\log N_{\text{C}}+N_{\text{near}}]t_{\text{fl}}t_{\text{lat}}\right)/T+[N_{\text{C}}+NM_{\text{near}}]t_{\text{bw}}t_{\text{lat}}})$ seconds per iteration using $M_{\max}\sim\sqrt{[N_{\text{C}}\log N_{\text{C}}+N_{\text{near}}]t_{\text{fl}}/(t_{\text{lat}}T)+[N_{\text{C}}+NM_{\text{near}}]t_{\text{bw}}/t_{\text{lat}}}$ processes. These costs should be contrasted to those of the pure MPI-AIM using $P=MT$ processes in Section 4.1, e.g., each MPI-AIM process requires a total of $O([N_{\text{C}}\log N_{\text{C}}+N_{\text{near}}]t_{\text{fl}}/(MT)+[MT+P_{\text{near}}]t_{\text{lat}}+[N_{\text{C}}+NP_{\text{near}}]t_{\text{bw}}/(MT))$ seconds per iteration that can be reduced at best to $O(\sqrt{[N_{\text{C}}\log N_{\text{C}}+N_{\text{near}}]t_{\text{fl}}t_{\text{lat}}+[N_{\text{C}}+NP_{\text{near}}]t_{\text{bw}}t_{\text{lat}}})$ seconds using $P_{\max}\sim\sqrt{[N_{\text{C}}\log N_{\text{C}}+N_{\text{near}}]t_{\text{fl}}/t_{\text{lat}}+[N_{\text{C}}+NP_{\text{near}}]t_{\text{bw}}/t_{\text{lat}}}$ processes. To sum up, the hybrid MPI/OpenMP approach can reduce the minimum time required for the matrix

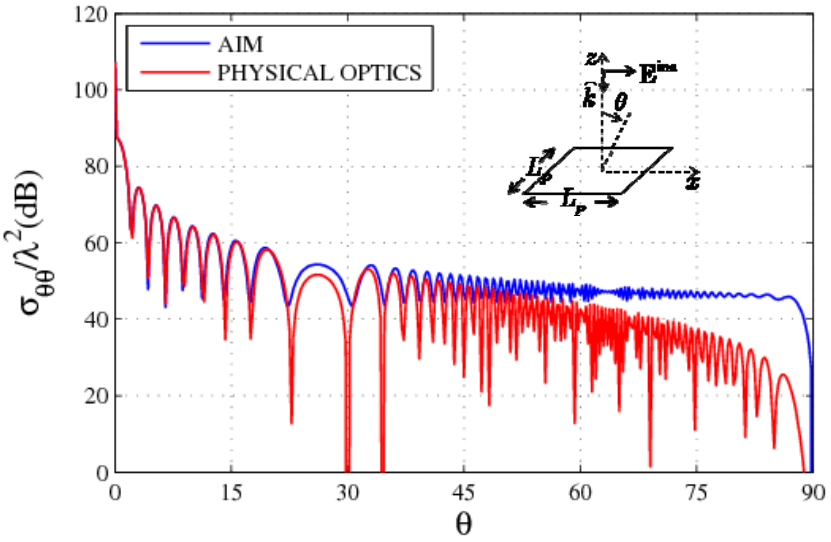
solve step by a factor of $\sqrt{T}$ compared to the pure MPI approach by using $M_{\max}T$ cores instead of $P_{\max}$ cores ($M_{\max}T = \sqrt{T}P_{\max}$ in this case.)

4.4 Numerical Results

This section presents numerical results using the benchmark spheres and plates described in Chapter 2. First, the accuracy of hybrid MPI/OpenMP-AIM is demonstrated. Then, the scalability of MPI-AIM and hybrid MPI/OpenMP-AIM are investigated.

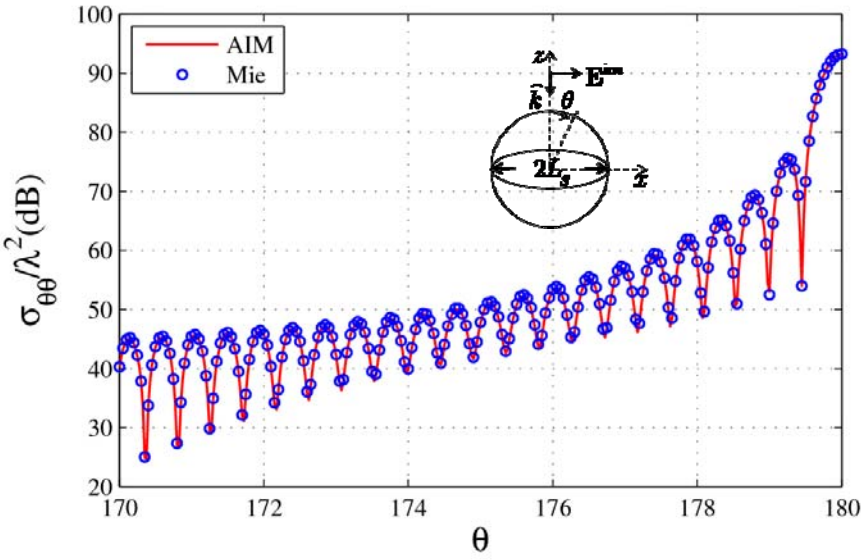
A. Accuracy Validation

To demonstrate the accuracy of the hybrid MPI/OpenMP-AIM, the largest benchmark plate $\left(L_P = 256\lambda\right)$ and sphere $\left(L_S = 64\lambda\right)$ are illuminated by an $\hat{x}$-polarized plane wave propagating toward $-\hat{z}$ direction. Fig. 4.7(a) and (b) compare the VV-polarized bistatic RCS of the plate and sphere to the (approximate) physical optics [39] and (exact) Mie series solutions, respectively. Good agreement is observed with reference results. Here, the AIM results are obtained by using the parameters given in Chapter 2 and by terminating the iterative solver when the relative residual error is less than 10^{-4}.



(a)

Figure 4.7: Bistatic RCS (VV) of the largest benchmark scatterers: (a) Plate $\left(L_P = 256\lambda\right)$ (b) Sphere $\left(L_S = 64\lambda\right)$.



(b)

Figure 4.7: Continued.

B. Scalability

First, the difference between scaling M and T are highlighted using the 3-D log-log-log plots introduced in Section 3.3. Fig. 4.8 shows the computational requirements for analyzing scattering from the $L_P = 8\lambda$ plate and $L_S = 4\lambda$ sphere along the (logarithmic) *z* axis versus M and T along the (logarithmic) *x* and *y* axes, respectively. These specific examples are simulated because they are large enough to be in the asymptotic complexity regime of AIM and small enough to observe both near-ideal scalability and scalability limitations using relatively small MT. Fig. 4.8(a), (c), and (e) on the left-hand side (Fig. 4.8(b), (d), and (f) on the right-hand side) show the wall-clock time during the matrix fill step, the average wall-clock time for one iteration during the matrix solve step, and the maximum memory per core during the entire analysis required by the MPI-AIM and hybrid MPI/OpenMP-AIM implementations for the plate (sphere) simulation, respectively. Similar observations to the MOM results can be made for AIM from these figures: (i) The matrix fill step of both implementations exhibit near-ideal scalability. (ii) The matrix solve step of hybrid MPI/OpenMP-AIM exhibits better strong

scalability in comparison to MPI-AIM. (iii) The memory requirement of hybrid MPI/OpenMP-AIM exhibits better strong memory scalability compared to the MPI-AIM.

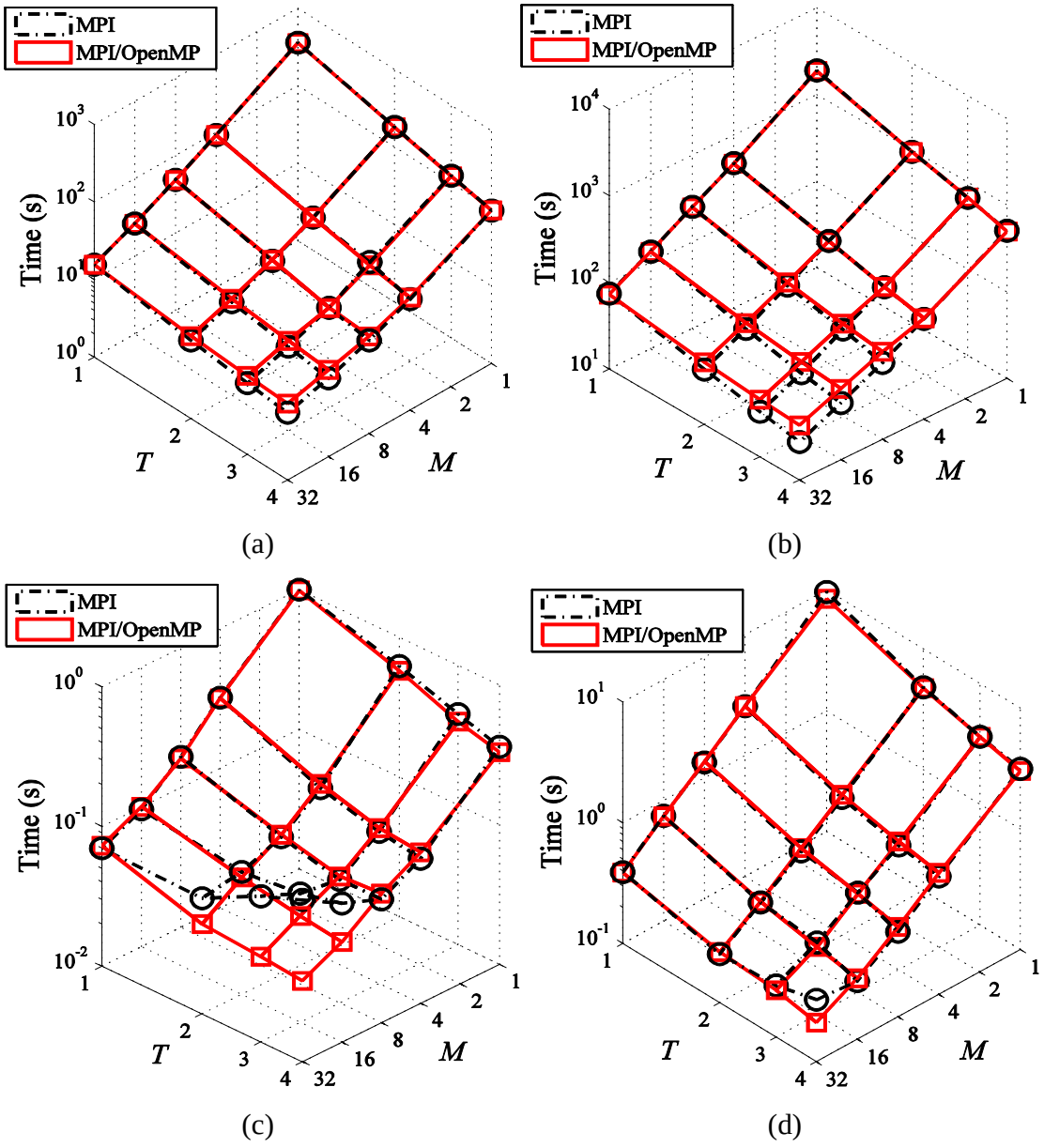
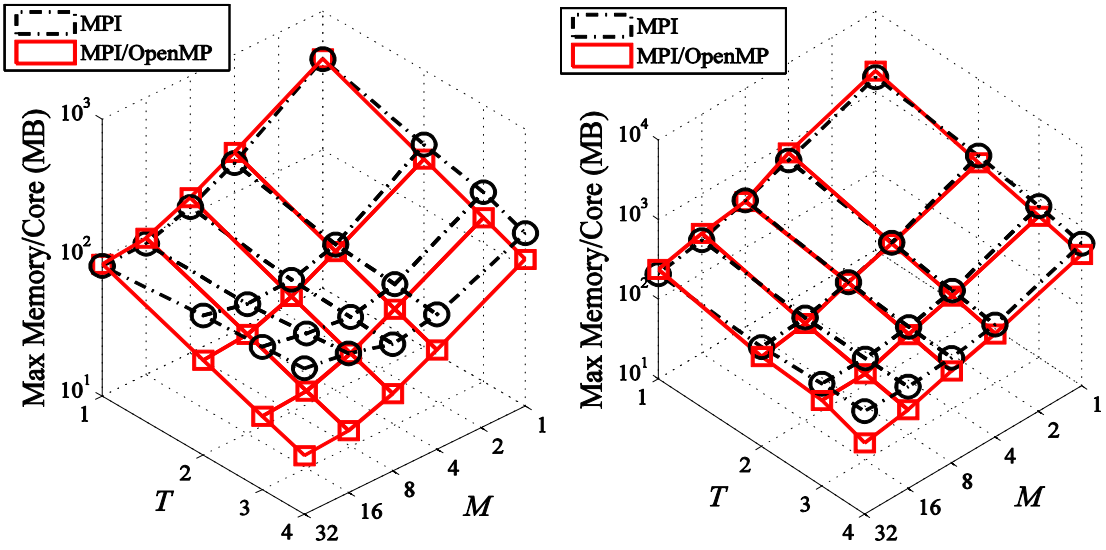


Figure 4.8: Computational requirements for analyzing scattering from an $L_P = 8\lambda$ plate (on the left-hand side) and an $L_S = 4\lambda$ sphere (on the right-hand side) using MPI-AIM and hybrid MPI/OpenMP-AIM. Wall clock time (matrix fill) for (a) plate and (b) sphere. Average wall clock time per iteration (matrix solve) for (c) plate and (d) sphere. Maximum memory needed per core for (e) plate and (f) sphere.



(e) (f)

Figure 4.8: Continued.

Next, more detailed observations are made by using 2-D log-log plots introduced in Section 3.3. Figs. 4.9 and 4.10 show the computational requirements for several of the benchmark plates and spheres along the (logarithmic) *y* axis versus the total number of active cores $P = MT$ along the (logarithmic) *x* axis, respectively. The data in the figures are obtained by varying M and fixing T. Fig. 4.9(a), (c), and (e) on the left-hand side (Fig. 4.9(b), (d), and (f) on the right-hand side) show the wall-clock time during the matrix fill step, the average wall-clock time for one iteration during matrix solve step, and the maximum memory among all cores during the entire analysis required by the MPI-AIM (hybrid MPI/OpenMP-AIM) implementation for different plate simulations, respectively. Fig. 4.10(a), (c), and (e) on the left-hand side (Fig. 4.10(b), (d), and (f) on the right-hand side) show the corresponding data for different sphere simulations.

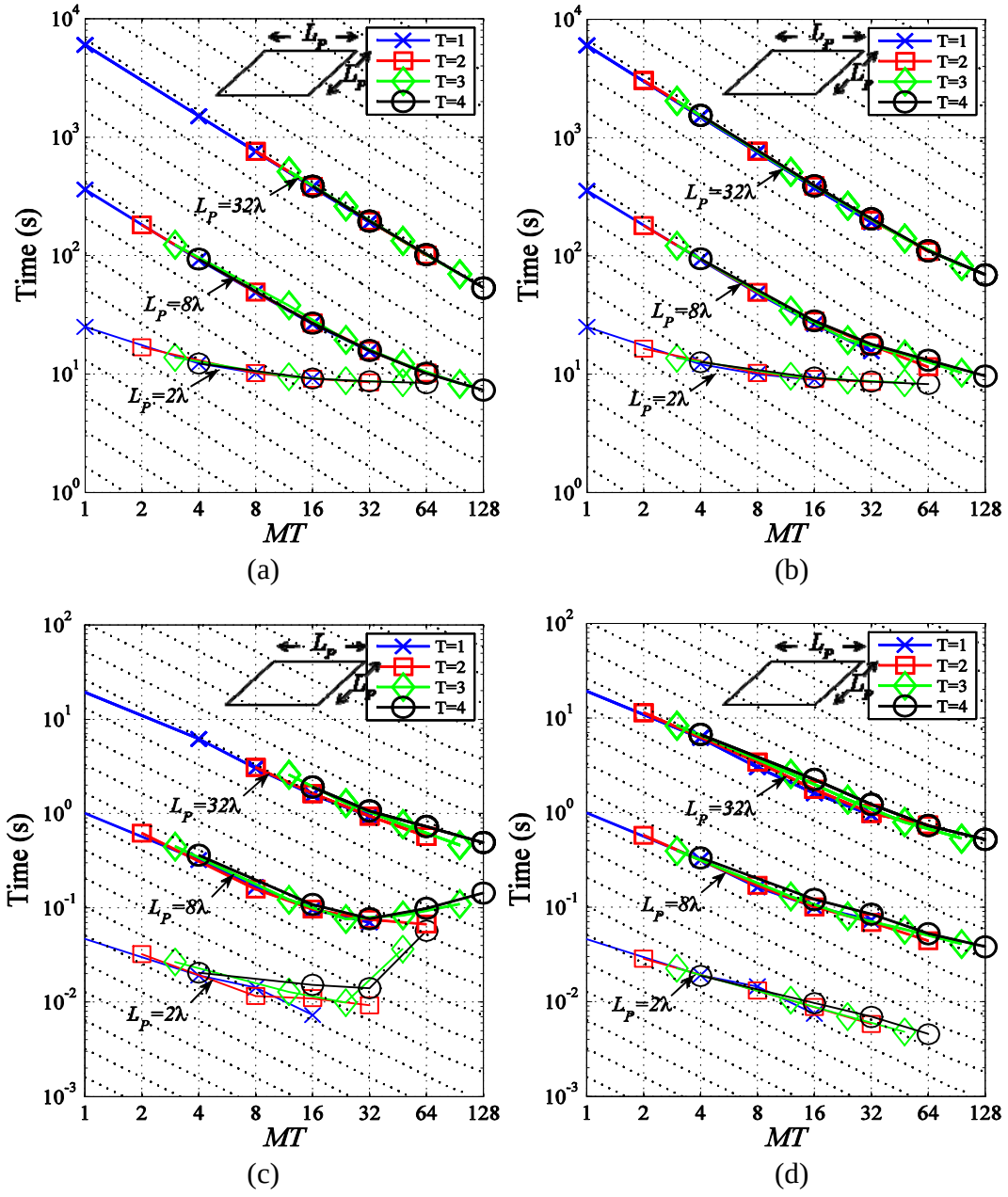
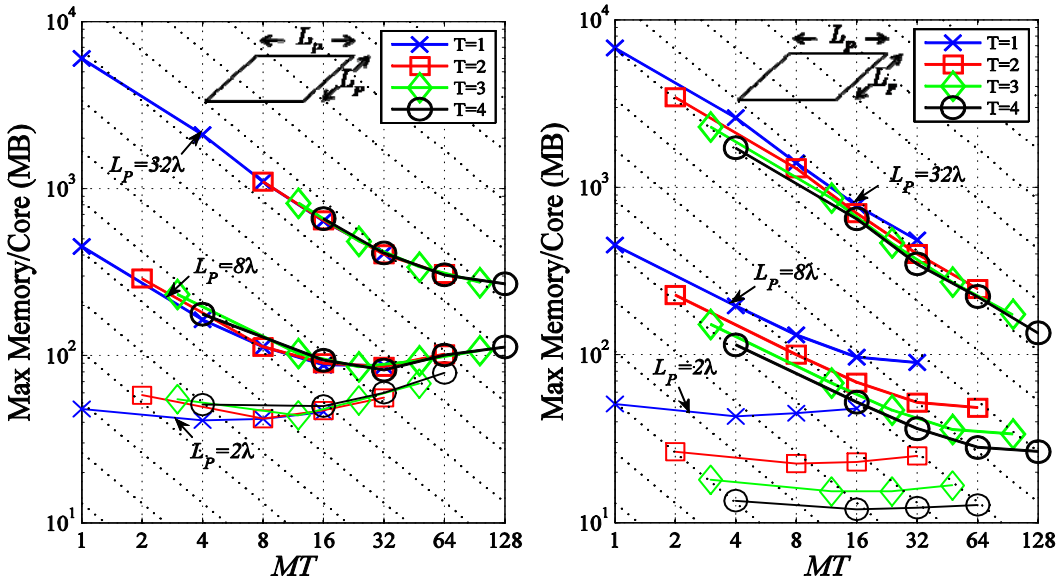


Figure 4.9: Computational requirements for analyzing scattering from $L_P = 2\lambda$, $L_P = 8\lambda$ and $L_P = 32\lambda$ plates using MPI-AIM (on the left-hand side) and hybrid MPI/OpenMP-AIM (on the right-hand side). Wall clock time (matrix fill) for (a) MPI-AIM and (b) hybrid MPI/OpenMP-AIM. Average wall clock time per iteration (matrix solve) for (c) MPI-AIM and (d) hybrid MPI/OpenMP-AIM. Maximum memory needed per core for (e) MPI-AIM and (f) hybrid MPI/OpenMP-AIM. Dashed lines are ideal scalability tangents.



(e) (f)

Figure 4.9: Continued.

Figs. 4.9(a)-(b) (Figs. 4.10(a)-(b)) show that the matrix fill step of both implementations exhibit near-ideal scalability for plates (spheres)[8]. It is observed that the memory and communication hierarchy of Ranger cluster can be ignored for the AIM matrix fill step because the time required for the matrix fill step does not depend on the particular choice of M and T[9] but is a function of the total number of cores $P = MT$.

Figs. 4.9(c)-(d) (Fig. 4.10(c)-(d)) show that the matrix solve step of the hybrid MPI/OpenMP-AIM is less sensitive to the choice of T for plates (spheres). Indeed, the more cores are active in each processor (the larger T is) the less scalable MPI-AIM becomes. Moreover, both parallelization approaches show limited scalability compared to their MOM counterparts in Figs. 3.7(c)-(d) (Figs. 3.8(c)-(d)) for the same number of unknowns as expected. Overall, the memory and communication hierarchy of Ranger

[8] The scalability of the matrix fill step for the smallest plate and sphere are observed to be limited. This unexpected result appears to be because "FFTW_MEASURE" flag is used in the implementation when creating FFTW plans. Further tests show that using "FFTW_ESTIMATE" flag during plan creation overcomes this limitation and the matrix fill time becomes ideally scalable even for the smallest scatterers.

[9] The matrix fill step for the $L_S = 16\lambda$ sphere for the hybrid MPI/OpenMP-AIM appears to be a function of T. The source of this anomaly is not clear at the time of the writing of this thesis.

cannot be ignored for the AIM matrix solve step, and the hybrid MPI/OpenMP-AIM better exploits this hierarchy to outperform the MPI-AIM during the matrix solve step.

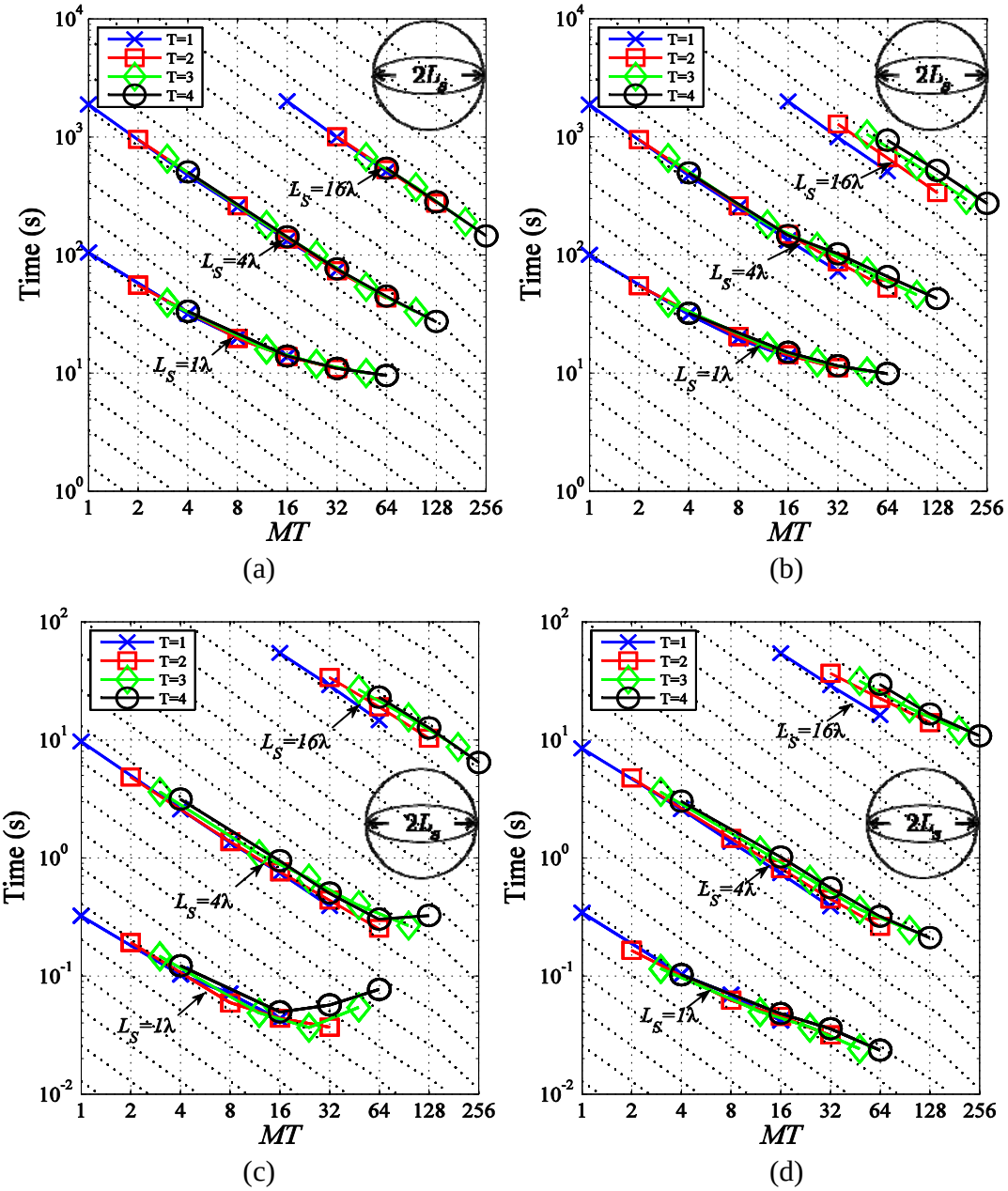


Figure 4.10: Computational requirements for analyzing scattering from $L_S = 1\lambda$, $L_S = 4\lambda$, and $L_S = 16\lambda$ spheres using MPI-AIM (on the left-hand side) and hybrid MPI/OpenMP-AIM (on the right-hand side). Wall clock time (matrix fill) for (a) MPI-AIM and (b) hybrid MPI/OpenMP-AIM. Average wall clock time per iteration (matrix solve) for (c) MPI-AIM and (d) hybrid MPI/OpenMP-AIM. Maximum memory needed per core (e) MPI-AIM and (f) hybrid MPI/OpenMP-AIM. Dashed lines are ideal scalability tangents.

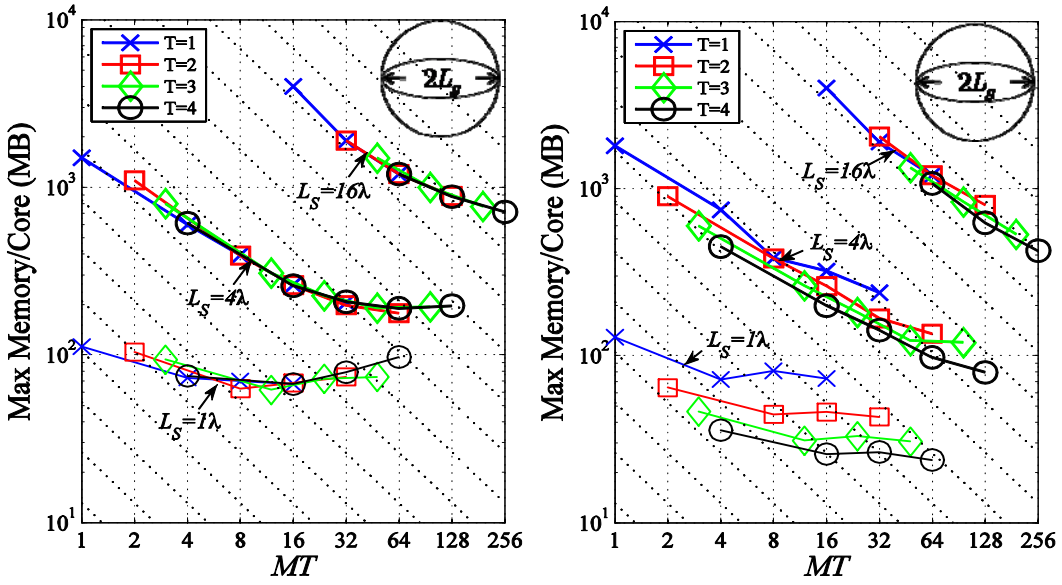


Figure 4.10: Continued.

Figs. 4.9(e)-(f) (Fig. 4.10(e)-(f)) show that the maximum memory requirement of the hybrid MPI/OpenMP-AIM implementation exhibits better scalability compared to the MPI-AIM. The memory requirement of both implementations falls down to a constant level as the total number of cores increases due to the same reasons as in Section 3.3.

Next, the weak scalability of the two implementations are compared by focusing on the matrix solve step, which is the main parallel scalability bottleneck. Fig. 4.11 shows the average wall-clock time required per iteration for the benchmark scatterers; the data in the figure are obtained by varying M and fixing T to the two extreme cases: $T=1$ (1 core is active per processor) and $T=4$ (all cores are active). Figs. 4.11(a) and (c) (Fig. 4.11(b) and (d)) show the results of all plate (sphere) simulations for $T=1$ and $T=4$, respectively.

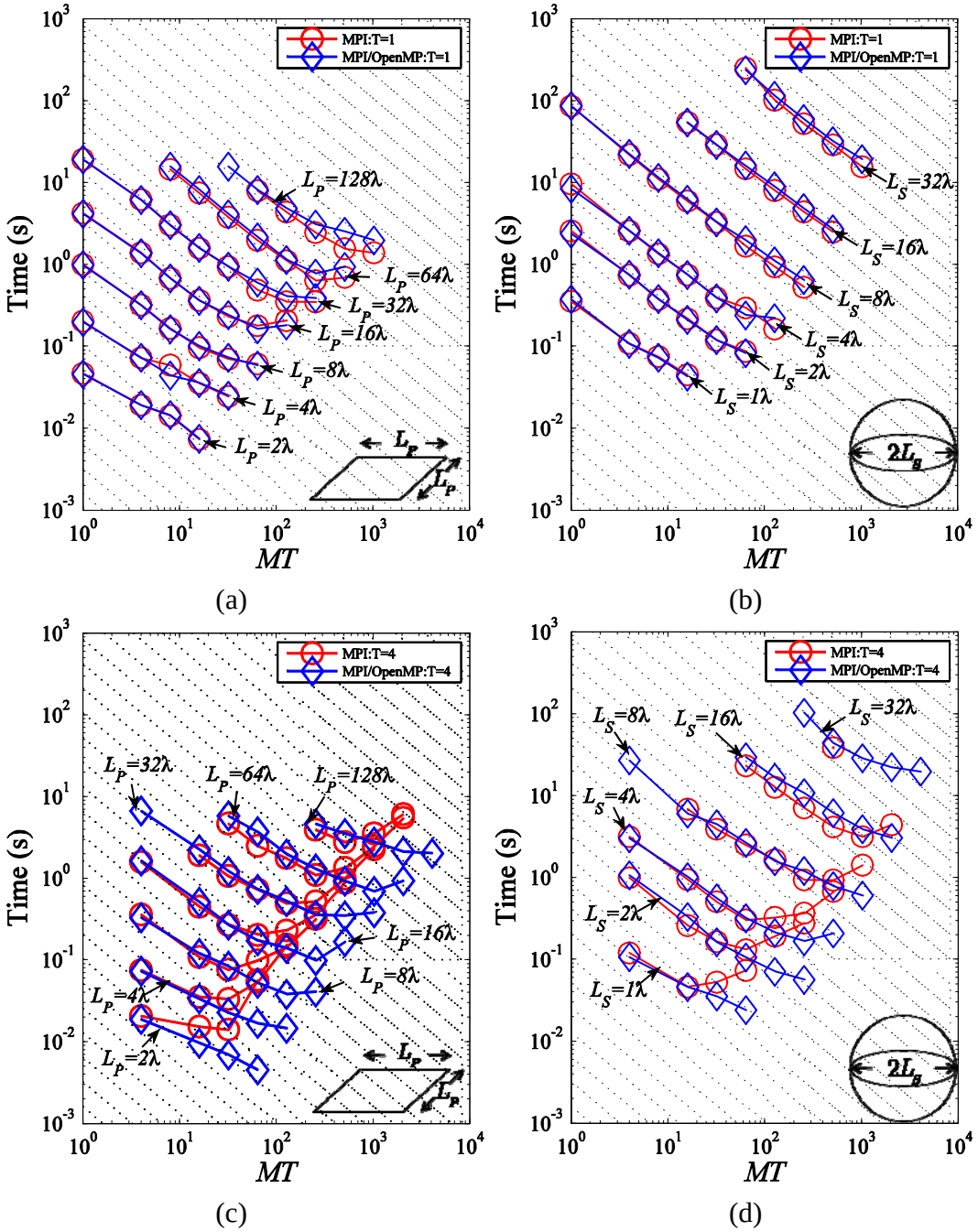
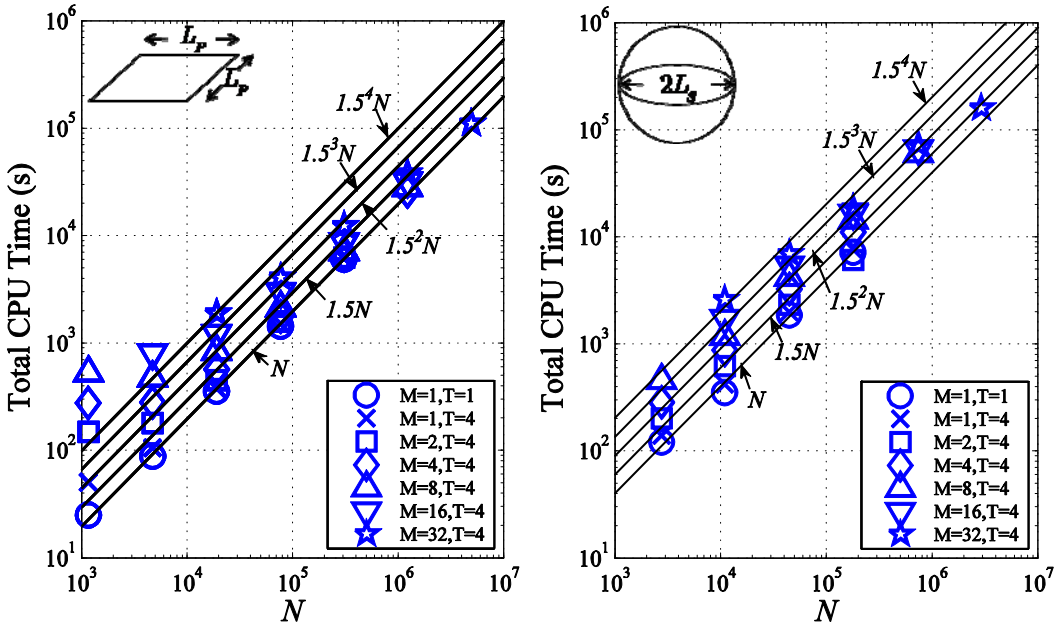


Figure 4.11: Average wall clock time per iteration (matrix solve) for plate (on the left-hand side) and sphere (on the right-hand side) simulations: (a) Plates and (b) spheres for $T = 1$ and (c) plates and (d) spheres for $T = 4$.

For the $T = 1$ case, the timing data for both implementations should theoretically be identical but the data in Figs. 4.11(a)-(b) show small differences, which are less than 5% in all cases except for the largest plate data (where the differences are less than 15%).

The small differences might reflect random variations in execution time from one simulation to the next or OpenMP overheads for executing one-threaded parallel regions. For the $T = 4$ case, the advantages of the hybrid MPI/OpenMP-AIM are clear (Figs. 4.11(c)-(d)). Fig. 4.11(c) noticeably shows that the $P_{\max}$ for MPI-AIM increases with the plate size. Indeed, the effectively usable number of cores for MPI-AIM doubles as the number of unknowns quadruples for the plates, which agrees with the analysis in Section 4.1 that $P_{\max} \sim \sqrt{[N_{\mathrm{C}} \log N_{\mathrm{C}} + N_{\mathrm{near}}]t_{\mathrm{fl}} / t_{\mathrm{lat}} + [N_{\mathrm{C}} + NP_{\mathrm{near}}]t_{\mathrm{bw}} / t_{\mathrm{lat}}}$ ($N_{\mathrm{C}} \sim N$ for quasi-planar geometries). Although less visible, the hybrid MPI/OpenMP-AIM results also follow the predicted trends.



(a) (b)

Figure 4.12: Computational complexity of the hybrid MPI/OpenMP-AIM for plates (on the left-hand side) and spheres (on the right-hand side) when using different number of cores. Total CPU time (matrix fill) for (a) plates and (b) spheres. Total CPU time per iteration (matrix solve) for (c) plates and (d) spheres. Total memory needed for (e) plates and (f) spheres.

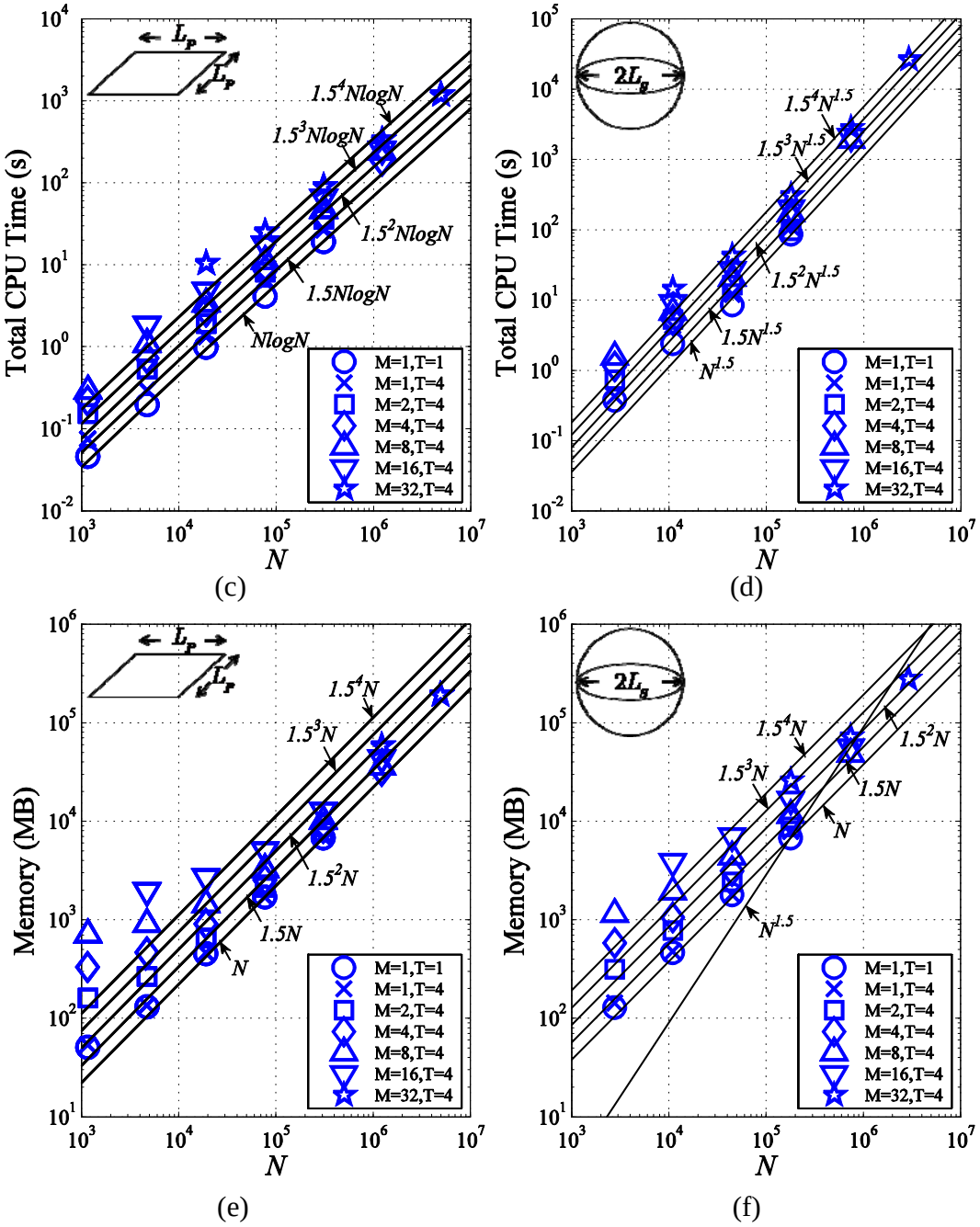


Figure 4.12: Continued.

Finally, the weak-scalability of the parallelization is investigated by quantifiying the impact of parallelization on the computational complexity of AIM. Figs. 4.12(a), (c), and (e) on the left-hand side (Figs. 4.12(b), (d), and (f) on the right-hand side) show the total computation time during the matrix fill step, the total computation time per iteration

during matrix solve step, and the total memory requirement of the hybrid MPI/OpenMP-AIM with respect to N for plates (spheres), respectively. The data points are obtained using the data in Fig. 4.11; i.e., the wall-clock time and memory requirement per core for different number of cores are multiplied with the corresponding number of cores to yield the total computation time and memory requirement, respectively. As a result, if ideal scalability was achieved, the different data points for a given N should coincide with each other.

Figs. 4.12(a), (c), and (e) (Figs. 4.12 (b), (d), and (f)) show that the matrix fill time, the matrix solve time per iteration, and the memory requirement of the hybrid MPI/OpenMP-AIM scales as $O(N)$, $O(N \log N)$, and $O(N)$ ($O(N)$, $O(N^{1.5})$, and $O(N^{1.5})$) for plates (spheres), respectively.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

This thesis presented a hybrid MPI/OpenMP technique for scalable parallelization of AIM on clusters of identical multi-core processors. The proposed hybrid MPI/OpenMP parallelization scheme is based on a nested 1-D slab decomposition of the 3-D auxiliary uniform grid and the associated AIM calculations. If M MPI processes processors and T OpenMP threads per process are used (a total of MT cores are active) then the scheme (i) partitions the uniform grid into M slabs in x dimension and assigns each slab to one MPI process (ii) further partitions each slab to T sub-slabs and assigns the associated computations to each thread.

It was shown that the parallel scalability of the MOM and AIM are limited by the communication time during the matrix solve step. Theoretical analysis and numerical studies on benchmark scatterers showed that (i) parallel scalability of MOM can be either latency or bandwidth limited depending on the cluster, whereas scalability of AIM is principally latency limited and (ii) the hybrid MPI/OpenMP parallelization improves the scalability of both MOM and AIM by reducing the impact of latency (which is achieved mainly through the communication of larger chunks of data using fewer messages). Moreover, the hybrid MPI/OpenMP-AIM does not replicate non-parallelized data structures among different cores of a processor; thus, its memory requirement is practically independent of T and exhibits better multi-core scalability than MPI-AIM.

Finally, it should be observed that the maximum number of cores participating in the nested 1-D slab decomposition is limited by the largest two dimensions of the 3-D auxiliary uniform grid, which limits the available memory and thus the problem size. A 2-D decomposition of the 3-D auxiliary uniform grid has the potential to increase the maximum number of cores that participate in the process and is currently being investigated.

REFERENCE

[1] R. F. Harrington, "Matrix methods for field problems," *Proc. IEEE*, vol. 55, pp. 136-149, Feb. 1967.

[2] W. C. Chew, *et al.*, *Fast and Efficient Algorithms in Computational Electromagnetics*: Norwood, MA: Artech House, 2001.

[3] W. C. Chew, *et al.*, "Fast solution methods in electromagnetics," *IEEE Trans. Antennas Propagat.*, vol. 45, pp. 533-543, Mar. 1997.

[4] V. Rokhlin, "Diagonal forms of translation operators for the Helmholtz equation in three dimensions," *Appl. Comp. Harmonic Anal.*, vol. 1, pp. 82-93, 1993.

[5] J. M. Song and W. C. Chew, "Multilevel fast multipole algorithm for solving combined field integral equations of electromagnetic scattering," *Microw. Opt. Tech. Lett.*, vol. 10, pp. 14-19, Sep. 1995.

[6] E. Bleszynski, *et al.*, "AIM: adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Sci.*, vol. 31, pp. 1225-1251, 1996.

[7] J. R. Phillips and J. K. White, "A precorrected-FFT method for electrostatic analysis of complicated 3-D structures," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 1059-1072, Oct. 1997.

[8] H. T. Anastassiu, *et al.*, "Scattering from relatively flat surfaces using the adaptive integral method," *Radio Sci.*, vol. 33, no. 10, pp. 7-16, 1998.

[9] A. E. Yılmaz, *et al.*, "Time domain adaptive integral method for surface integral eqautions," *IEEE Trans. Antennas Propagat.*, vol. 52, no. 10, pp. 2692-2708, Oct. 2004.

[10] E. J. Lu and D. I. Okunbor, "A massively parallel fast multipole algorithm in three dimensions," presented at the 5th IEEE Int. Symp. Parallel Distrib. Process., 1996.

[11] J. M. Song and W. C. Chew, "The fast illinois solver code: Requirements and scaling properties," *IEEE Comput. Sci. Eng.*, vol. 5, pp. 19-23, 1998.

[12] S. Velamparambil and W. C. Chew, "Analysis and performance of a distributed memory multilevel fast multipole algorithm," *IEEE Trans. Antennas Propag.*, vol. 54, pp. 2719-2727, Aug. 2007.

[13] Ö. Ergül and L. Gürel, "Efficient Parallelization of the Multilevel Fast Multipole Algorithm for the Solution of Large-Scale Scattering Problems," *IEEE Trans. Antennas Propag.*, vol. 56, pp. 2335-2345, Aug. 2007.

[14] A. Buchau, *et al.*, "Parallelization of a Fast Multipole Boundary Element Method with Cluster OpenMP," *IEEE Trans. Magnetics*, vol. 44, pp. 1338-1341, Jun. 2008.

[15] M. A. Heroux, "Design issues for numerical libraries on scalable multicore architectures," presented at the J. Phys.: Conf. Ser., 2008.

[16] F. Cappello, *et al.*, "Understanding performance of SMP clusters running MPI programs," *Fut. Gen. Comp. Sys.*, vol. 17, pp. 711-720, Apr. 2001.

[17] D. S. Henty, "Performance of hybrid message-passing and shared-memory parallel-ism for discrete element modeling," in *Proc. ACM/IEEE Supercomputing*, 2000, p. 10.

[18] T. Q. Viet, *et al.*, "Construction of hybrid MPI/OpenMP solutions for SMP

clusters," *IPSJ Trans. Adv. Comp. Sys.*, vol. 46, pp. 25-37, 2005.
[19] G. H. R. Rabenseifner, and G. Jost, "Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes," in *Proc. of the 17th Euromicro Conference on Parallel, Distributed and Network-based Processing*, Weimar, Germany, 2009.
[20] M. F. Su, *et al.*, "A novel FDTD application featuring OpenMP-MPI hybrid parallelization," in *Proc. Int. Conf. Parallel Processing (ICPP)*, 2005, pp. 373-379.
[21] M. C. Hughes and M. A. Stuchly, "Hybrid parallel finite difference time domain simulation of nanoscale optical phenomena," in *Proc. IEEE/ACES Wireless Comm. Appl. Comp. Electromagn.*, 2005, pp. 132-135.
[22] H. Mahawar and V. Sarin, "Parallel software for inductance extraction," in *Proc. Int. Conf. Parallel Process. (ICPP)*, 2004, pp. 380-386.
[23] X. Wang and V. Jandhyala, "Enhanced hybrid MPI-OpenMP parallel electromagnetic simulations based on low-rank compressions," in *Proc. IEEE Int. Symp. Electromagn. Compat.*, Aug. 2008, pp. 1-5.
[24] *http://www.top500.org/system/9257*.
[25] J. R. Mautz and R. F. Harrington, "A combined-source solution for radiation and scattering from a perfectly conducting body," *IEEE Trans. Antennas Propag.*, vol. 27, pp. 445-454, July 1979.
[26] P. Ylä-Oijala, *et al.*, "Surface integral equation method for general composite metallic and dielectric structures with junctions," *Progress Electromagn. Research*, vol. 52, pp. 81-108, 2005.
[27] S. M. Rao, *et al.*, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Trans. Antennas Propagat.*, vol. 30, pp. 409-418, May 1982.
[28] G. J. Burke and A. J. Poggio, "Numerical Electromagnetic Code (NEC) - method of moments," Naval Ocean Systems Center, San Diego, CA, Tech. Document 116. July 1977.
[29] S. L. Ray and A. F. Peterson, "Error and convergence in numerical implementations of the conjugate gradient method," *IEEE Trans. Antennas Propagat.*, vol. 36, Dec. 1988.
[30] R. W. Freund, "A Transpose-Free Quasi-Minimal Residual Algorithm for Non-Hermitian Linear Systems," *SIAM J. Sci. Compt.*, vol. 14, pp. 470-482, Mar. 1993.
[31] D. B. Davidson, "Parallel matrix solvers for moment method codes for MIMD computers," *ACES Journal*, vol. 8, no.2, pp. 144-175, 1993.
[32] Q. G. L. Chai, and D. Panda., "Understanding the impact of multi-core architecture in cluster computing: a case study with Intel dual-core system," in *Proceedings of CCGrid 2007*, Rio de Janeiro, Brazil, May 2007.
[33] A. H. L. Chai, and D. K. Panda, "Designing High Performance and Scalable MPI Intra-node Communication," in *The IEEE International Conference on Cluster Computing*, 2006.
[34] S. R. Alam, *et al.*, "Characterization of Scientific Workloads on Systems with multicore processors," in *International Symposium on Workload Characterization*, 2006.
[35] B. Tu, *et al.*, "Multi-core aware optimization for MPI collectives," in *IEEE International Conference on Cluster Computing*, Tsukuba, Japan, 2008.

[36] A. V. Oppenheim and R. W.Schafer, *Discrete-time signal processing*, 2nd ed.: Prentice Hall, 1999.
[37] *http://www.fftw.org.*
[38] S. Williamsa, *et al.*, "Optimization of sparse matrix–vector multiplication on emerging multicore platforms," *Parallel Computing*, vol. 35, pp. 178-194, Mar. 2009.
[39] C. A. Balanis, *Advanced Engineering Electromagnetics*. New York.: John Wiley & Sons, Publishers, Inc., .